

# **Diseño e implementación de una herramienta para la visualización de código para alumnos de asignaturas de introducción a la programación en ingenierías. (ID2017/003)**

Memoria de resultados

Convocatoria de Innovación Docente – Curso 2017-2018



**VNiVERSIDAD  
D SALAMANCA**

CAMPUS OF INTERNATIONAL EXCELLENCE

Alberto López Barriuso (Coordinador)  
Daniel Hernández de la Iglesia  
Álvaro Lozano Murciego  
Juan Francisco de Paz Santana

Departamento de Informática y Automática  
Universidad de Salamanca - Facultad de Ciencias  
Plaza de la Merced, s/n 37008 Salamanca



## Contenido

Contenido .....	3
1. Datos del proyecto .....	5
2. Introducción .....	6
3. Objetivos.....	7
4. Desarrollo del proyecto .....	8
Estructura de la guía .....	9
5. Resultados logrados .....	9
6. Conclusiones.....	14

## Ilustraciones

Ilustración 1 Página principal de la web.....	8
Ilustración 2 Herramienta de visualización de código interactiva. ....	8
Ilustración 3 Relación de ficheros de la página web .....	10
Ilustración 4 1. ¡Programación en C- Introducción! .....	10
Ilustración 5 Hola Mundo!.....	10
Ilustración 6 Variables y tipos.....	11
Ilustración 7 Arrays.....	11
Ilustración 8 Arrays multidimensionales .....	11
Ilustración 9 Cadenas de caracteres.....	12
Ilustración 10 Bucles.....	12
Ilustración 11 Funciones.....	12
Ilustración 12 static .....	13
Ilustración 13 Punteros.....	13
Ilustración 14 Estructuras .....	13
Ilustración 15 Parámetros por referencia .....	14
Ilustración 16 Memoria dinámica.....	14



Diseño e implementación de una herramienta para la visualización de código para alumnos de asignaturas de introducción a la programación en ingenierías. (ID2017/003)

## 1. Datos del proyecto

**Título:** Diseño e implementación de una herramienta para la visualización de código para alumnos de asignaturas de introducción a la programación en ingenierías

**Referencia:** (ID2017/003)

**Cuantía de la subvención:** 0€

**Coordinador del proyecto:** Alberto López Barriuso

**Organismo:** Universidad de Salamanca

**Centro:** Facultad de Ciencias

**Investigadores que forman el equipo:**

Alberto López Barriuso (Coordinador)

Daniel Hernández de la Iglesia

Álvaro Lozano Murciego

Juan Francisco de Paz Santana

**Duración:** Octubre 2017 – Junio 2018

Diseño e implementación de una herramienta para la visualización de código para alumnos de asignaturas de introducción a la programación en ingenierías. (ID2017/003)

## 2. Introducción

La experiencia como profesores en diferentes asignaturas de programación, nos ha ayudado a detectar ciertos aspectos de estas asignaturas cuyo aprendizaje puede resultar costoso a los alumnos. El comprender cómo el código fuente de un programa, siendo éste un texto estático, es asignado a un proceso dinámico como es la ejecución de un programa, no es trivial para gran parte del alumnado. Del mismo modo, la comprensión de conceptos básicos de estas asignaturas, como lo son los punteros, el ámbito de las variables o el funcionamiento de la gestión dinámica de la memoria resulta generalmente complejo para los alumnos.

Tradicionalmente, la explicación de la ejecución de los programas se apoya en material didáctico como diapositivas o en presentaciones Power Point, las cuales, además de requerir de un gran esfuerzo para su preparación, en ocasiones pueden resultar confusas para el alumnado. El uso de herramientas de visualización de código puede contribuir en gran medida a mejorar la calidad de la enseñanza en las asignaturas de programación, facilitando a los alumnos la comprensión de los conceptos más complejos y permitiendo a los profesores generar nuevo material didáctico de gran calidad sin necesidad de emplear una gran cantidad de tiempo, como era necesario utilizar en la elaboración del material habitual en estas asignaturas.

Por estas razones, este proyecto tiene como objetivo principal la implantación de un servidor que dé soporte a una herramienta de visualización de código. Esta herramienta pretende ayudar a los alumnos a superar una barrera fundamental a la hora de aprender programación: comprender qué ocurre en el ordenador cuando se ejecuta cada línea del código fuente de un programa. La plataforma, basada en la herramienta pythontutor, será accesible a través de un sitio web, permitiendo su uso sin que sea necesario instalar ningún componente adicional tales como extensiones o plugins.

Diseño e implementación de una herramienta para la visualización de código para alumnos de asignaturas de introducción a la programación en ingenierías. (ID2017/003)

### 3. Objetivos

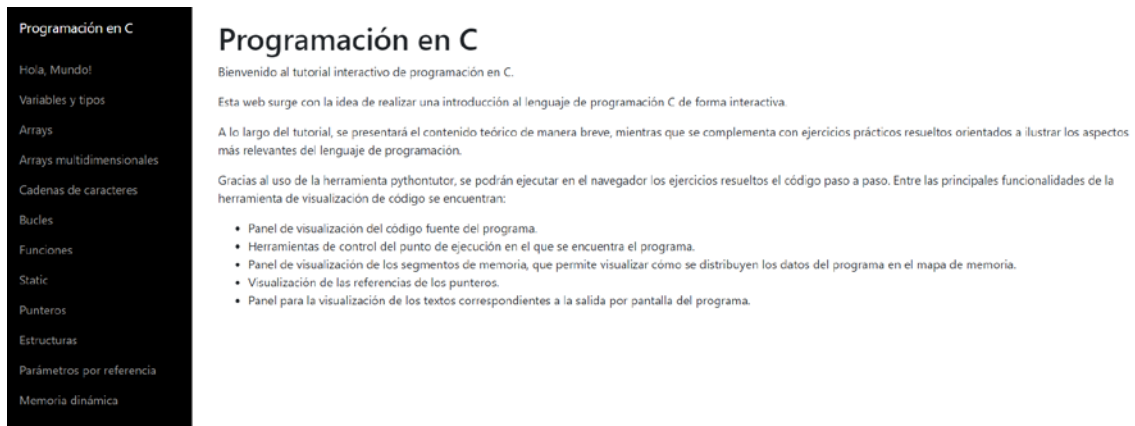
El proyecto presentado pretende innovar en una de las áreas de conocimiento más importantes para los estudiantes de una titulación de Ingeniería, la programación. Se trata de una materia que para la mayoría de estudiantes resulta completamente desconocida y que para muchos marcará su futuro laboral una vez finalizados sus estudios. Esta materia, para todos aquellos alumnos que la abordan por primera vez, suele resultar confusa y difícil de estudiar. Los conceptos que se abordan en los inicios de esta asignatura (como variables, funciones, bucles, etc) suelen ser explicados por los profesores mediante apuntes y transparencias que resultan a menudo poco didácticas. Por todo ello, gracias a la herramienta que se pretende desarrollar en este proyecto, se esperan conseguir las siguientes innovaciones:

- En primer lugar, obtener una innovación en la manera en la que los profesores imparten sus asignaturas de programación en clase.
- En esta misma línea, obtener una innovación en la manera en la que los profesores realizan sus apuntes y presentaciones para que los sus alumnos estudien y trabajen la asignatura.
- Otra de las principales innovaciones es la manera en la que los alumnos podrán estudiar y trabajar con la programación, a través de un entorno visual, analizando las instrucciones paso a paso.

Diseño e implementación de una herramienta para la visualización de código para alumnos de asignaturas de introducción a la programación en ingenierías. (ID2017/003)

#### 4. Desarrollo del proyecto

Con el fin de crear una guía que fuera interactiva para todos aquellos alumnos y usuarios que desearan consultarla, se ha desarrollado una página web con diferentes secciones. En cada una de ellas, se introducen una serie de conceptos teóricos acerca del lenguaje de programación C y a continuación se presentan pequeños programas que apoyan la explicación teórica.



**Programación en C**

Hola, Mundo!

Variables y tipos

Arrays

Arrays multidimensionales

Cadenas de caracteres

Bucles

Funciones

Static

Punteros

Estructuras

Parámetros por referencia

Memoria dinámica

## Programación en C

Bienvenido al tutorial interactivo de programación en C.

Esta web surge con la idea de realizar una introducción al lenguaje de programación C de forma interactiva.

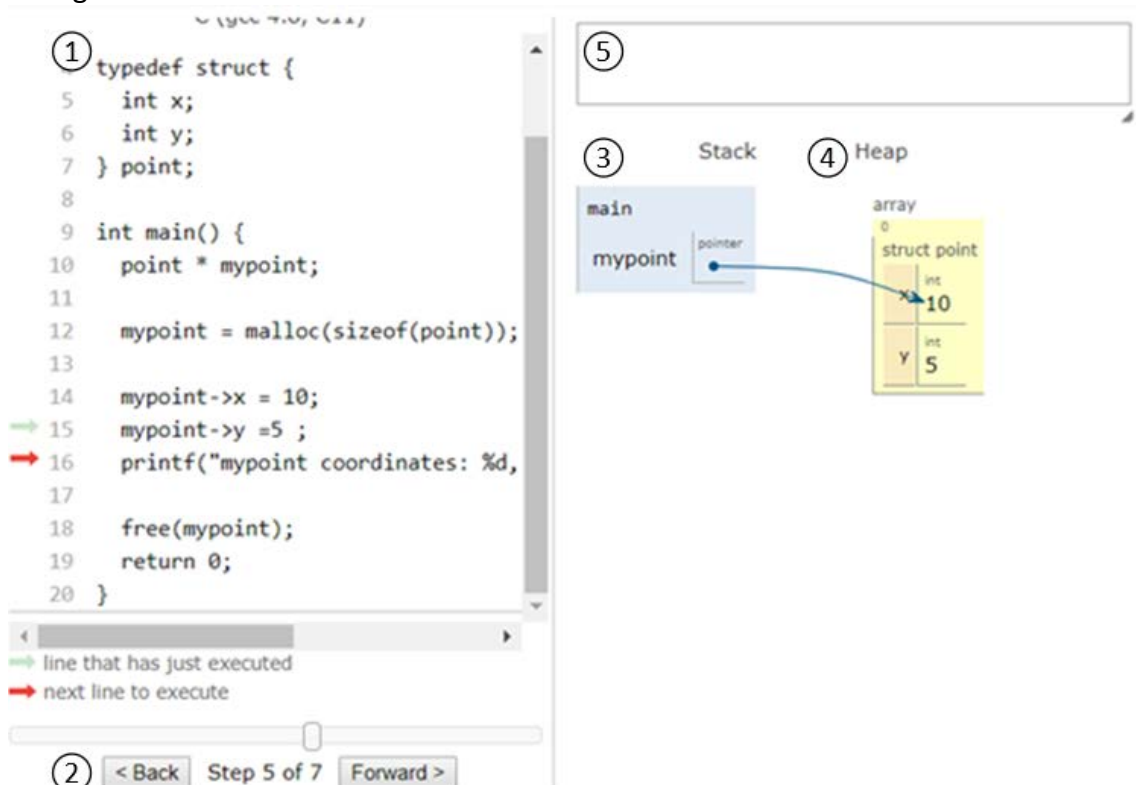
A lo largo del tutorial, se presentará el contenido teórico de manera breve, mientras que se complementa con ejercicios prácticos resueltos orientados a ilustrar los aspectos más relevantes del lenguaje de programación.

Gracias al uso de la herramienta pythontutor, se podrán ejecutar en el navegador los ejercicios resueltos el código paso a paso. Entre las principales funcionalidades de la herramienta de visualización de código se encuentran:

- Panel de visualización del código fuente del programa.
- Herramientas de control del punto de ejecución en el que se encuentra el programa.
- Panel de visualización de los segmentos de memoria, que permite visualizar cómo se distribuyen los datos del programa en el mapa de memoria.
- Visualización de las referencias de los punteros.
- Panel para la visualización de los textos correspondientes a la salida por pantalla del programa.

Ilustración 1 Página principal de la web

Para favorecer la asimilación de los contenidos, estos programas se muestran mediante una herramienta de visualización de código interactiva (Ilustración 2), que dispone de los siguientes elementos:



```
1 typedef struct {
2     int x;
3     int y;
4 } point;
5
6 int main() {
7     point * mypoint;
8
9     mypoint = malloc(sizeof(point));
10
11     mypoint->x = 10;
12     mypoint->y = 5 ;
13     printf("mypoint coordinates: %d,
14
15     free(mypoint);
16     return 0;
17 }
```

5

3 Stack 4 Heap

main  
mypoint pointer

array  
0  
struct point  
int 10  
int 5  
y

2 < Back Step 5 of 7 Forward >

→ line that has just executed  
→ next line to execute

Ilustración 2 Herramienta de visualización de código interactiva.



Diseño e implementación de una herramienta para la visualización de código para alumnos de asignaturas de introducción a la programación en ingenierías. (ID2017/003)

1. Pantalla de código fuente: La pantalla del código fuente muestra el programa que se está visualizando. Una flecha roja en el margen izquierdo apunta a la siguiente línea que se ejecutará (línea 7 en este ejemplo). Una flecha verde clara apunta a la línea que acaba de ejecutarse, lo que ayuda a los usuarios a rastrear el flujo de control no contiguo (por ejemplo, llamadas a funciones).
2. Avanzar/retroceder en la siguiente línea de Código a ejecutar: un slider y un pequeño texto indican el punto de ejecución actual que se está visualizando (en este ejemplo, el paso 5 de 7). Cada punto representa una sola línea ejecutada. El usuario puede hacer clic o arrastrar el ratón sobre la barra deslizadora para saltar a un punto particular o usar los botones de navegación estilo para avanzar y retroceder sobre las líneas ejecutadas.
3. Panel de marcos: El panel de marcos muestra las variables globales y marcos que enmarcan las variables de la pila en el punto de ejecución actual, con la pila creciendo hacia abajo. Cada cuadro muestra el nombre de la función y una lista de variables locales. El cuadro actualmente activo se resalta en azul.
4. Panel heap: este panel muestra aquellas variables reservadas dinámicamente y que, por lo tanto, se encuentran almacenadas en el montículo.
5. Ver la salida por pantalla del programa.

## Estructura de la guía

Se ha estructurado la guía en diferentes unidades con el fin de agrupar los elementos que la componen de una manera sencilla e intuitiva para el alumno. La estructura es la siguiente:

- 1. Programación en C- Introducción**
- 2. ¡Hola, Mundo!**
- 3. Variables y tipos**
- 4. Arrays**
- 5. Arrays multidimensionales**
- 6. Cadenas de caracteres**
- 7. Bucles**
- 8. Funciones**
- 9. Static**
- 10. Punteros**
- 11. Estructuras**
- 12. Parámetros por referencia**
- 13. Memoria dinámica**

## 5. Resultados logrados

Los ficheros de la página web guía didáctica resultante de este proyecto de innovación docente se encuentran adjuntos en la entrega. Existen un total de 12 secciones y una página principal.

# Diseño e implementación de una herramienta para la visualización de código para alumnos de asignaturas de introducción a la programación en ingenierías. (ID2017/003)

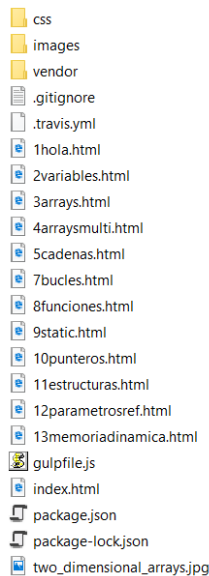


Ilustración 3 Relación de ficheros de la página web

La página web es accesible a través de la siguiente dirección:

<https://albarriuso.github.io/visual-code.github.io/index.html>

A continuación, se muestran capturas de cada una de las secciones desarrolladas:

## 1. Programación en C- Introducción



Ilustración 4 1. ¡Programación en C- Introducción!

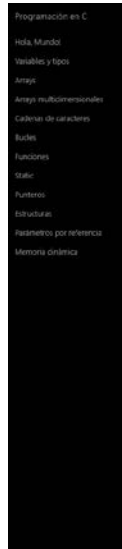
## 2. ¡Hola, Mundo!



Ilustración 5 Hola Mundo!

# Diseño e implementación de una herramienta para la visualización de código para alumnos de asignaturas de introducción a la programación en ingenierías. (ID2017/003)

## 3. Variables y tipos



### Variables y tipos

#### Tipos de datos

El lenguaje de programación C tiene varios tipos de variables, pero los más habituales son:

- **Enteros** - números enteros, positivos y negativos. Definidos mediante `char`, `int`, `short`, `long` o `long long`.
- **Enteros sin signo** - números enteros que sólo pueden ser positivos. Se pueden definir mediante los tipos de dato `unsigned char`, `unsigned int`, `unsigned short`, `unsigned long` o `unsigned long long`.
- **Números en coma flotante** - números reales. Definidos usando `float` and `double`.
- **Estructuras** - veán explicación más adelante, en la sección de estructuras.

Los diferentes tipos de variables definen sus límites. Un `char` puede tener un rango de -128 a 127, mientras que un `long` puede variar de -2,147,483,648 a 2,147,483,647 (`long` y otros tipos de datos numéricos pueden tener otro rango en diferentes computadores, por ejemplo - de -9,223,372,036,854,775,808 a 9,223,372,036,854,775,807 en una computadora de 64 bits).

Se debe tener en cuenta que C no cuenta con un tipo de dato booleano. Generalmente, se define usando la siguiente notación:

```
#define TRUE 1
```

```
#define FALSE 0
```

```
#define TRUE 1
```

#### Definiendo variables

Para los números, generalmente empleamos el tipo de dato `int`, que es un número entero en el tamaño de una "palabra", el tamaño de número predeterminado de la máquina en la que se compiló el programa. En la mayoría de los computadores, es un número de 32 bits, lo que significa que el número puede oscilar entre -2,147,483,648 y 2,147,483,647.

Para definir las variables `foo` y `bar` necesitamos usar la siguiente sintaxis:

```
int foo;
```

```
int bar = 1;
```

Ahora, podemos hacer algunas matemáticas. Suponiendo que `a`, `b`, `c`, `d` y `e` son variables, simplemente podemos usar los operadores más, menos y multiplicación en la siguiente notación, y asignar un nuevo valor a `a`:

Ilustración 6 Variables y tipos

## 4. Arrays



### Arrays

Los arrays son variables especiales que pueden contener más de un valor usando la misma variable, empleando un índice. Los arrays se definen mediante una sintaxis muy sencilla:

```
int numeros[10];
```

```
int numeros[10];
```

```
int numeros[10];
```

```
int numeros[10];
```

```
int numeros[10];
```

```
int numeros[10];
```

```
int numeros[10];
```

```
int numeros[10];
```

```
int numeros[10];
```

```
int numeros[10];
```

```
int numeros[10];
```

```
int numeros[10];
```

```
int numeros[10];
```

```
int numeros[10];
```

```
int numeros[10];
```

```
int numeros[10];
```

```
int numeros[10];
```

```
int numeros[10];
```

```
int numeros[10];
```

```
int numeros[10];
```

```
int numeros[10];
```

```
int numeros[10];
```

```
int numeros[10];
```

```
int numeros[10];
```

```
int numeros[10];
```

```
int numeros[10];
```

```
int numeros[10];
```

```
int numeros[10];
```

```
int numeros[10];
```

```
int numeros[10];
```

```
int numeros[10];
```

```
int numeros[10];
```

```
int numeros[10];
```

```
int numeros[10];
```

```
int numeros[10];
```

```
int numeros[10];
```

```
int numeros[10];
```

```
int numeros[10];
```

```
int numeros[10];
```

```
int numeros[10];
```

```
int numeros[10];
```

```
int numeros[10];
```

```
int numeros[10];
```

```
int numeros[10];
```

```
int numeros[10];
```

```
int numeros[10];
```

```
int numeros[10];
```

```
int numeros[10];
```

```
int numeros[10];
```

```
int numeros[10];
```

```
int numeros[10];
```

```
int numeros[10];
```

```
int numeros[10];
```

```
int numeros[10];
```

```
int numeros[10];
```

```
int numeros[10];
```

```
int numeros[10];
```

```
int numeros[10];
```

```
int numeros[10];
```

Ilustración 7 Arrays

## 5. Arrays multidimensionales



### Arrays multidimensionales

En la última sección, cubrimos los aspectos generales de los arrays y cómo funcionan. Sin embargo, C dispone de otro tipo de arrays: los arrays multidimensionales. A continuación se muestra la forma genérica de declarar un array multidimensional:

```
tipo nombre[tamaño1][tamaño2]...[tamañoN];
```

A continuación, se muestra un ejemplo de declaración básica de un array multidimensional:

```
int foo[10][10];
```

```
int foo[10][10];
```

```
int foo[10][10];
```

```
int foo[10][10];
```

```
int foo[10][10];
```

```
int foo[10][10];
```

```
int foo[10][10];
```

```
int foo[10][10];
```

```
int foo[10][10];
```

```
int foo[10][10];
```

```
int foo[10][10];
```

```
int foo[10][10];
```

```
int foo[10][10];
```

```
int foo[10][10];
```

```
int foo[10][10];
```

```
int foo[10][10];
```

```
int foo[10][10];
```

```
int foo[10][10];
```

```
int foo[10][10];
```

```
int foo[10][10];
```

```
int foo[10][10];
```

```
int foo[10][10];
```

```
int foo[10][10];
```

```
int foo[10][10];
```

```
int foo[10][10];
```

```
int foo[10][10];
```

```
int foo[10][10];
```

```
int foo[10][10];
```

```
int foo[10][10];
```

```
int foo[10][10];
```

```
int foo[10][10];
```

```
int foo[10][10];
```

```
int foo[10][10];
```

```
int foo[10][10];
```

```
int foo[10][10];
```

```
int foo[10][10];
```

```
int foo[10][10];
```

```
int foo[10][10];
```

```
int foo[10][10];
```

#### Arrays Bidimensionales

La forma más simple de array multidimensional es el array bidimensional. Un array bidimensional es prácticamente una lista de matrices unidimensionales. Para declarar un array bidimensional de enteros de tamaño [M][N], se debería realizar de la siguiente manera:

```
int nombreArray[M][N];
```

Donde tipo puede ser cualquier tipo de C (`int`, `char`, `long`, `long long`, `double`, etc.) nombreArray será un identificador válido en C o variable. Un array bidimensional se puede considerar como una tabla que tendrá [M] número de filas y [N] número de columnas. Un array bidimensional, que contiene tres filas y cuatro columnas, se conceptualizar así:

	Columna 0	Columna 1	Columna 2	Columna 3
Row 0	4	5	6	7
Row 1	8	9	10	11
Row 2	12	13	14	15

En este sentido, cada elemento del array, a está identificado de la siguiente forma: `a[i][j]`, donde 'i' es el nombre del array, 'i' y 'j' son los índices que lo identifican de manera única en 'a'.

A la hora de declarar un array de la siguiente manera, no es necesario poner un valor [i][j].

```
char nombreArray[M][N] = {  
    { 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', ' ' },  
    { 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z', ' ' }  
};
```

ya que el compilador sabrá que hay dos dimensiones, pero si es necesario un valor [i][j].

#### Inicializar arrays bidimensionales

Se pueden utilizar arrays multidimensionales especificando valores [i][j] entre corchetes para cada fila. A continuación se puede ver un vector con 3 filas y cada fila tiene 4 columnas. Para hacerlo más fácil, se puede omitir el 'j' y mantenerlo en blanco, aún así funcionará.

```
int a[3][4] = {  
    { 1, 2, 3, 4 }, /* inicializadores para la fila 0 (omitido por 'j' */  
    { 5, 6, 7, 8 }, /* inicializadores para la fila 1 (omitido por 'j' */  
    { 9, 10, 11, 12 } /* inicializadores para la fila 2 (omitido por 'j' */  
};
```

Los corchetes internos, que indican la 'fila deseada', son opcionales. La siguiente inicialización sería equivalente a la del ejemplo anterior:

```
int a[3][4] = {1,2,3,4,5,6,7,8,9,10,11,12};
```

#### Accediendo a elementos en arrays bidimensionales

Se accede a un elemento en una matriz bidimensional utilizando los subíndices, es decir, el índice de fila y el índice de columna del array. Por ejemplo:

```
int val = a[1][2];
```

Ilustración 8 Arrays multidimensionales

# Diseño e implementación de una herramienta para la visualización de código para alumnos de asignaturas de introducción a la programación en ingenierías. (ID2017/003)

## 6. Cadenas de caracteres

**Cadenas de caracteres**

Las cadenas en C son en realidad arrays de caracteres. Aunque los punteros en C están en un tema explicado más adelante, utilizaremos punteros a un conjunto de caracteres para definir cadenas simples, de la siguiente manera:

```
char * nombre = "John Smith";
```

Este método crea una cadena que sólo podemos usar para leer. Si deseamos definir una cadena que pueda ser manipulada, necesitaremos definirla como una matriz de caracteres local:

```
char nombre[] = "John Smith";
```

Esta notación es diferente porque asigna una variable de tipo array para que podamos manipularla. La notación de los corchetes vacíos [] le dice al compilador que calcule el tamaño de la matriz automáticamente. De hecho, esto es lo mismo que asignarlo explícitamente, agregando uno a la longitud de la cadena:

```
char nombre[] = "John Smith";  
/* es lo mismo que */  
char nombre[11] = "John Smith";
```

La razón por la que necesitamos añadir uno a la longitud de la cadena, aunque la cadena "John Smith" tiene exactamente 10 caracteres, es para indicar la terminación de la cadena mediante un carácter especial (igual a 0) que indica el final de la cadena. El final de la cadena está marcado porque el programa no conoce la longitud de la cadena, únicamente la conocerá de acuerdo con el código.

**Formateo de cadenas de caracteres con printf**

Podemos usar el comando `printf` para formatear una cadena junto con otras cadenas, de la siguiente manera:

```
char * nombre = "John Smith";  
int edad = 12;  
  
/* imprimir "John Smith tiene 12 años." */  
printf("Yo (%s) soy (%d) años", nombre, edad);
```

Se debe tener en cuenta que al imprimir cadenas, debemos agregar un carácter de nueva línea (`\n`) para que nuestra declaración `printf` se imprima en una nueva línea.

**Longitud de una cadena**

La función `strlen` devuelve la longitud de la cadena que se debe pasar como argumento:

```
char * nombre = "John Smith";  
printf("Yo (%s) tengo (%d) años", nombre, strlen(nombre));
```

**Comparación de cadenas**

La función `strcmp` compara dos cadenas, devolviendo el número 0 si son iguales, o un número diferente si son diferentes. Los argumentos son las dos cadenas que se compararán y la longitud máxima de comparación. También hay una versión insegura de esta función llamada `strcpy`, pero no se recomienda su uso. Por ejemplo:

```
char * nombre = "John";  
  
if (strcmp(nombre, "John") == 0) {  
    printf("¡Hola, John!\n");  
} else {  
    printf("No eres John.\n");  
}
```

**Concatenación de cadenas**

Ilustración 9 Cadenas de caracteres

## 7. Bucles

**Bucles for**

Los bucles for son directos. Brindan la posibilidad de crear un bucle (un bloque de código que se ejecuta varias veces). Para los bucles for se requiere un iterador, usualmente denotado como `i`.

Los bucles for proveen la siguiente funcionalidad:

- Inicializa la variable del iterador usando un valor inicial
- Comprueba si el iterador ha alcanzado su valor final
- Incrementa el iterador

Por ejemplo, si deseamos iterar en un bloque de código 10 veces, escribiremos:

```
int i;  
for (i = 0; i < 10; i++) {  
    printf("Hola", i);  
}
```

Este bloque imprimirá los números del 0 al 9 (10 números en total).

Los bucles for permiten iterar en los valores de un array. Por ejemplo, si quisiéramos sumar todos los valores de un array, usaríamos el iterador `i` como un índice del array:

```
int array[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };  
int sum = 0;  
int i;  
  
for (i = 0; i < 10; i++) {  
    sum += array[i];  
}
```

`/* sum should contain: 55 = 0(1) + ... + 9(1) */  
printf("La suma del array es %d", sum);`

**Ejemplo**

Calcular el factorial (multiplicación de todos los elementos desde `array[0]` hasta `array[9]`, ambos incluidos), de la variable `array`.

Ilustración 10 Bucles

## 8. Funciones

**Funciones**

Las funciones C son simples, pero debido a cómo funciona C, el poder de las funciones es un poco limitado.

- Las funciones reciben una cantidad fija o variable de argumentos.
- Las funciones sólo pueden devolver un valor o no devolver ningún valor.

En C, los argumentos se copian por valor a las funciones, lo que significa que no podemos cambiar los argumentos para afectar su valor fuera de la función. Para hacer eso, debemos usar punteros, que se enseñan más adelante.

Las funciones se definen con la siguiente sintaxis:

```
int fact(int bar) {  
    /* hacer algo */  
    return bar * 2;  
}  
  
int main() {  
    fact();  
}
```

La función `foo` que definimos recibe un argumento `bar`. La función recibe un número entero, lo multiplica por dos y devuelve el resultado.

Para ejecutar la función `foo` con el valor 1 para el argumento `bar`, usamos la siguiente sintaxis:

```
fact();
```

En C, las funciones se deben definir antes de que se utilicen en el código. Pueden declararse primero y luego implementarse usando un archivo de cabecera o al principio del archivo C, o pueden implementarse en el orden en que se usan (menos preferible).

La forma correcta de usar funciones es la siguiente:

```
/* declaración de la función */  
int fact(int bar);  
  
int main() {  
    /* llamada a la función foo desde main */  
    printf("the value of foo is %d", fact());  
}
```

```
int fact(int bar) {  
    return bar * 2;  
}
```

También podemos crear funciones que no devuelvan un valor usando la palabra clave `void`:

```
void foo() {  
    /* hacer algo y no devolver un valor */  
}
```

```
int main() {  
    foo();  
    fact();  
}
```

**Ejemplo**

Ilustración 11 Funciones

# Diseño e implementación de una herramienta para la visualización de código para alumnos de asignaturas de introducción a la programación en ingenierías. (ID2017/003)

## 9. Static

**Static**

`static` es una palabra clave en el lenguaje de programación C. Se puede usar con variables y funciones.

Por defecto, las variables son locales para el ámbito en el que están definidas. Las variables se pueden declarar como estáticas para aumentar su alcance hasta el archivo que las contiene. Como resultado, se puede acceder a estas variables en cualquier lugar dentro de un archivo.

Considera la siguiente situación, queremos contar los corredores que participan en una carrera:

```
C (gcc 4.6, C11)
1 #include <stdio.h>
2 int corredor() {
3     int count = 0;
4     count++;
5     return count;
6 }
7
8 int main()
9 {
10    printf("Id ", corredor());
11    printf("Id ", corredor());
12    return 0;
13 }
```

Print output (Step: lower right corner to resize)

Stack Heap

main

Step 1 of 11 Forward

Ilustración 12 static

## 10. Punteros

**Punteros**

Los punteros también son variables y juegan un papel muy importante en el lenguaje de programación C. Se usan para diferentes cuestiones, tales como:

- Cadenas de caracteres
- Asignación de memoria dinámica
- Pasar argumentos de funciones por referencia
- Construir estructuras de datos complejas
- Apuntar a funciones
- Creación de estructuras de datos especiales (listas, árboles, etc.)

Entre otras:

**¿Qué es un puntero?**

Un puntero es esencialmente una variable entera simple que contiene una dirección de memoria que apunta a un valor, en lugar de contener el valor real en sí.

La memoria de la computadora es almacenada secuencial de datos, y un puntero apunta a una parte específica de la memoria. Nuestro programa puede usar punteros de tal manera que los punteros apuntan a una gran cantidad de memoria, dependiendo de cuánto necesitamos leer a partir de ese momento.

**Cadenas como punteros**

Ya hemos probado las cadenas anteriormente, pero ahora podemos profundizar un poco más y comprender que son las cadenas en C realmente.

La siguiente línea:

```
char * cadena = "Hola";
```

hace tres cosas:

- Define una variable local llamada `cadena`, que es un puntero a un carácter.
- Indica que la cadena "Hola" aparece en algún lugar de la memoria del programa (después de compilarse y ejecutarse).
- Inicializa el argumento `cadena` de forma que apunte a la zona de memoria donde se encuentra el carácter 'H' (al cual sigue el resto de la cadena en la memoria).

Si tratamos de acceder a la variable `cadena` como una matriz, funcionará y devolverá el valor ordinal del carácter 'H', ya que la variable `cadena` en realidad apunta exactamente al comienzo de la cadena.

Como sabemos que la memoria es secuencial, podemos suponer que si avanzamos en la memoria al siguiente carácter, recibiremos el siguiente letra de la cadena. Hasta que lleguemos al final de la cadena, marcada con un terminador nulo (el carácter cuyo valor ordinal es 0 anidado como `\0`).

**Desreferencia**

Desreferencia la operación de referirse a donde señala el puntero, en lugar de la dirección de memoria. Ya estamos usando la desreferenciación en vectores. El operador corchete `[]` (por ejemplo, accede al primer elemento del vector). Dado que los vectores son en realidad punteros, acceder al primer elemento del vector es lo mismo que desreferenciar un puntero. La desreferenciación de un puntero se realiza utilizando el operador asterisco `*`.

Si queremos crear un array que apunte a una variable diferente en nuestra pila, podemos escribir el siguiente código:

```
int *apuntar_una_variable_diferente() {
    int x = 1;
    int *puntero;
    apuntar_una_variable_diferente, y apunta a la variable x utilizando el operador &.
    *puntero = *x;
    printf("El valor de x es %d", *x);
    printf("El valor de *x es %d", *puntero);
}
```

Utilizamos el operador `&` para apuntar a la variable `x`, que es nuestro objetivo.

Ilustración 13 Punteros

## 11. Estructuras

**Estructuras**

Las estructuras en C son variables que contienen varias variables nombradas en su interior. Se emplean para:

- Serialización de datos
- Pasar múltiples argumentos dentro y fuera de las funciones a través de un único argumento
- Estructuras de datos, como listas enlazadas, árboles binarios, etc.

El ejemplo más típico del uso de estructuras son los **puntos**, una entidad única que contiene dos variables: `x` y `y`. Definamos un punto en un espacio bidimensional:

```
typedef struct {
    int x;
    int y;
} punto;
```

Ahora, pasamos a definir un nuevo punto. Supongamos que la función `suma` recibe un punto y lo muestra por pantalla. Sin estructuras, su uso requeriría dos argumentos, cada uno para cada coordenada:

```
void suma(punto p) {
    printf("x: %d y: %d\n", p.x, p.y);
}
```

Utilizando estructuras, podemos pasar un único argumento de tipo punto:

```
void suma(punto p) {
    printf("x: %d y: %d\n", p.x, p.y);
}
```

Para acceder a las variables del punto, usamos el operador punto `.`:

```
suma(punto p) {
    printf("x: %d y: %d\n", p.x, p.y);
}
```

**Typedefs**

Typedefs nos permiten asignar un nombre diferente a un tipo de dato, lo que puede ser útil cuando se trabajamos con estructuras y punteros. En este caso, nos gustaría deshacernos de la definición larga de una estructura de puntos. Podemos usar la siguiente sintaxis para eliminar la palabra clave `struct` cada vez que queremos definir un nuevo punto:

```
typedef struct {
    int x;
    int y;
} punto;
```

Esto nos permite definir un nuevo punto:

```
punto p;
```

Las estructuras también pueden contener punteros, lo que les permite mantener cadenas o punteros a otras estructuras también. Por ejemplo, podemos definir una estructura de vehículo de la siguiente manera:

```
typedef struct {
    char * marca;
    int velocidad;
} vehiculo;
```

Como la marca es un puntero de char, el tipo de vehículo puede contener una cadena (que, en este caso, indica la marca del vehículo).

**Ejemplo**

Defina una nueva estructura de datos, llamada "persona", que contenga una cadena (puntero a char) llamada `nombre`, y un entero llamado `edad`.

Ilustración 14 Estructuras

# Diseño e implementación de una herramienta para la visualización de código para alumnos de asignaturas de introducción a la programación en ingenierías. (ID2017/003)

## 12. Parámetros por referencia

**Parámetros por referencia**

Los argumentos de una función se pasan por valor, lo que significa que se copian dentro y fuera de las funciones. Pero, ¿y si copiamos punteros en lugar de los propios valores? Esto nos permitirá dar a las funciones funciones el control sobre las variables y estructuras de las funciones primitivas, y no solo una copia de ellas.

Digamos que queremos escribir una función que incremente un número por uno, llamado `suma`. Esto no funcionará:

```
void suma(int n) {
    n++;
}

int x;
printf("antes: %d\n", x);
main();
printf("después: %d\n", x);
```

sin embargo, esto sí funcionará:

```
void suma(int * n) {
    (*n)++;
}

int x;
printf("antes: %d\n", x);
main();
printf("después: %d\n", x);
```

La diferencia es que la segunda versión de `suma` recibe un puntero a la variable `x` como argumento, y luego puede manipularlo, porque sabe dónde está en la memoria.

Oscurece que cuando se llama a la función `suma`, debemos pasar una referencia a la variable `x`, y no la variable en sí misma, esto se hace para que la función conozca la dirección de la variable y no reciba una copia de la propia variable.

**Punteros a estructuras**

Digamos que queremos crear una función que mueva un punto hacia adelante en ambas direcciones, `x` e `y`. Llamada `move`. En lugar de enviar dos punteros, ahora podemos enviar a la función un solo puntero de la estructura de puntos:

```
void move(punto * p) {
    (*p).x++;
    (*p).y++;
}
```

sin embargo, si deseamos desreferenciar una estructura y acceder a uno de sus miembros internos, tenemos una sintaxis abreviada para eso, porque esta operación se usa ampliamente en las estructuras de datos. Podemos reescribir esta función usando la siguiente sintaxis:

```
void move(punto * p) {
    p->x++;
    p->y++;
}
```

**Ejemplo**

Ilustración 15 Parámetros por referencia

## 13. Memoria dinámica

**Memoria dinámica**

La asignación dinámica de memoria es un tema muy importante en C que permite construir estructuras de datos complejas como listas vinculadas. Asignar memoria dinámicamente nos ayuda a almacenar datos sin conocer inicialmente el tamaño de los datos en el momento en que escribimos el programa.

Para asignar dinámicamente un espacio de memoria, debemos tener un puntero listo, que almacenará la ubicación de la memoria asignada. Podemos acceder a la memoria que nos fue asignada usando esa misma puntero, y podemos usar ese puntero para liberar la memoria que tenemos, una vez que terminemos de usarla.

Supongamos que queremos asignar dinámicamente una estructura persona que se define de la siguiente manera:

```
typedef struct {
    int id;
    char * nombre;
} persona;
```

Para asignar una nueva persona en el argumento `persona`, usamos la siguiente sintaxis:

```
persona * nuevaPersona = malloc(sizeof(persona));
```

Esto le dice al compilador que queremos asignar dinámicamente la memoria suficiente para almacenar una estructura de tipo persona en la memoria, y luego devolver un puntero a los datos recién asignados.

Para acceder a los miembros de la persona, podemos usar la notación `->`:

```
nuevaPersona->nombre = "Juan";
nuevaPersona->edad = 21;
```

Una vez que hayamos terminado de usar la estructura asignada dinámicamente, podemos liberarla usando `free`:

```
free(nuevaPersona);
```

Se debe tener en cuenta que la función `free` no elimina la variable `nuevaPersona` en sí misma, simplemente libera los datos a los que apunta. La variable `nuevaPersona` aún apunta a algún lugar de la memoria, pero después de llamar a `free` ya no podemos acceder a esa área. No debemos usar ese puntero nuevamente hasta que le asignemos nuevos datos.

**Ejemplo**

Ilustración 16 Memoria dinámica

## 6. Conclusiones

Como se hacía referencia en los objetivos del proyecto, la utilización de herramientas de visualización de código ha demostrado ser tremendamente útil en la docencia. Tanto para la parte en la que el profesor debe enseñar conceptos abstractos como son las estructuras de datos, cómo se almacenan en memoria las variables y, sobre todo, en el desarrollo de la ejecución de un programa: cómo es posible ver paso a paso con esta herramienta qué sucede y cómo se comporta realmente el ordenador en su interior cuando se ejecuta un programa.

Esto hace la tarea de la enseñanza mucho más sencilla ya que el alumno puede ver, paso a paso, lo que sucede. Por parte del alumno, este tipo de herramientas son especialmente útiles. En muchas ocasiones los alumnos que llegan al primer curso de las distintas carreras jamás han tenido contacto con la programación. Esto hace que asignaturas de este curso, relacionadas con la informática, en las que se imparte programación de algún lenguaje (Python, C, Java etc.) para muchos alumnos resulten demasiado arduas. Los alumnos ven la programación como una disciplina poco clara y poco intuitiva que no comprenden con facilidad, debido a la gran cantidad de

Diseño e implementación de una herramienta para la visualización de código para alumnos de asignaturas de introducción a la programación en ingenierías. (ID2017/003)

abstracción que existe en los conceptos que se imparten. Gracias a estas herramientas, se logra suavizar la curva de aprendizaje, ya que se puede observar qué es lo que realmente pasa y los propios alumnos pueden probar los programas que escriben en ellas.

Con la implantación de esta herramienta se pretenden conseguir mejoras tanto en la impartición de docencia por parte de los profesores, haciendo que sus explicaciones lleguen de manera más efectiva a los alumnos. Por otra parte, conseguir que los alumnos adquieran conceptos relacionados con la programación mucho más rápido y pierdan el miedo que tienen en muchos casos cuando se enfrentan por primera vez a la programación de cualquier lenguaje.

La guía de la herramienta para el profesorado permitirá que los docentes de las asignaturas mencionadas con anterioridad se beneficien de las ventajas que ofrece utilizar una herramienta de visualización de código para impartir sus clases (tanto teóricas como prácticas). Esta guía será pública y al alcance de cualquier docente que decida incluir la herramienta para la impartición de la asignatura. La colección de ejemplos ayudará a alumnos y profesores a realizar ejercicios para adquirir destrezas en la programación de manera más rápida y eficaz. Estos ejemplos estarán disponibles de manera pública para que cualquier alumno/docente pueda utilizarlos en cualquier momento.