

ESCUELA POLITÉCNICA SUPERIOR
INFORMÁTICA – CURSO 2024-2025
PRÁCTICA 1. INTRODUCCIÓN A LA PROGRAMACIÓN EN C

PRERREQUISITOS

- Para el correcto desarrollo de esta práctica, el alumno ANTES DE LA SESIÓN DE PRÁCTICA, imprimirá la práctica en papel pues **deberá llevarla a la sesión**, la leerá con atención y estudiará el contenido de la parte teórica de la misma.
- Además, se recomienda que descargue, instale e intente habituarse al entorno de programación que se utilizará durante todo el curso: **Zinjal** (se encuentra disponible para Windows en Enseñanza Virtual, y para Linux y OS X en la web oficial <http://zinjai.sourceforge.net/> . Se recomienda a los usuarios de Windows descargar la versión que está disponible en Enseñanza Virtual)

OBJETIVOS

Esta práctica tiene como objetivo la iniciación en la programación en lenguaje C:

- Se introduce el concepto de programa y programación.
- Se presentan los primeros conceptos del lenguaje C: variables, sus tipos y la instrucción de asignación.
- Se introduce el concepto de análisis de código y depuración.
- Se describe el entorno de programación Zinjal que se usará en el presente curso académico.

Al finalizar esta práctica, el alumno debe ser capaz de:

- Usar correctamente el entorno de programación.
- Crear nuevos ficheros de código fuente en C, abrir ficheros existentes, editar e identificar código correctamente, compilar, identificar errores de compilación y ejecutar el código compilado.
- Escribir programas en C que manejen variables y observar su funcionamiento usando el depurador.

PROGRAMACIÓN IMPERATIVA

Los circuitos electrónicos de un computador pueden reconocer datos e instrucciones que están constituidos por conjuntos de unos y ceros. Es lo que se conoce por lenguaje máquina. El lenguaje máquina es difícil de manejar y raramente puede contener operaciones más complejas que realizar operaciones aritméticas con dos números, mover datos de una parte de la memoria a otra y saltar a una instrucción si se cumple una condición. Una forma de enfrentarse a este problema es crear un nuevo lenguaje, llamado **lenguaje de programación de alto nivel** (como es el lenguaje C), que sea más fácil de utilizar por un humano, pero no para el computador. Para que el computador pueda ejecutarlos es necesario traducirlos a su propio lenguaje máquina. Ésta es una tarea que realiza un programa especial llamado **compilador**. Los programas, que son secuencias de instrucciones, escritos en lenguaje de alto nivel se almacenan con una extensión determinada: para el caso del lenguaje de alto nivel que estudiamos en este curso, el lenguaje C, la extensión de los ficheros es **.c**. La tarea del compilador se suele descomponer en dos etapas:

- En la primera etapa traduce el programa del fichero **.c** al lenguaje ensamblador produciendo el fichero **.o**. El lenguaje ensamblador es un lenguaje más próximo al del computador.
- En la segunda etapa genera el programa ejecutable en lenguaje máquina, que es el fichero con extensión **.exe** (en sistemas operativos Windows)

Como se ha descrito previamente, un programa es una secuencia de instrucciones que describe cómo ejecutar cierta tarea. Las instrucciones de los programas se ejecutan –ordinariamente- de modo **secuencial**, es decir, cada una a continuación de la anterior. Recientemente se está haciendo un gran esfuerzo en desarrollar programas paralelos, es decir, programas que se pueden ejecutar simultáneamente en varios procesadores. La programación paralela es mucho más complicada que la secuencial y no se hará referencia a ella en este curso.

PROGRAMACIÓN EN LENGUAJE C

Función main

Un programa C se divide en *funciones* (que se llaman *procedimientos o rutinas* en otros lenguajes). Cada función ejecuta una parte de un programa, pudiendo recibir unos valores (de entrada) y pudiendo devolver otro (de salida). El programador puede hacer cuantas funciones necesite y darles los nombres que quiera, pero siempre debe haber una función **main** (en inglés significa “principal”) que es la función por la que comienza la ejecución del programa. En las primeras prácticas ÚNICAMENTE trabajaremos con esta función principal.

La sintaxis de la función main es:

```
int main ()
{
    // Esto es un comentario y no se trata como sentencia
    /* Esto también
    es un comentario y no se traduce a código máquina */
    lista de instrucciones;
}
```

Todo lo que hay entre las llaves { ... } es el cuerpo de la función. El cuerpo está formado por una lista de instrucciones, separadas por el carácter ';'. Las instrucciones se procesan de forma secuencial.

Variables

Las variables nos permiten almacenar datos los cuales pueden ser de distinto formato numérico y tamaño. Hay diversos tipos de variables en C: enteras, caracteres, flotantes, vectores... Es necesario declararlas al principio (después de `main () { ... }`) con el objetivo de indicar al compilador que reserve un espacio determinado en memoria. La longitud en *bytes* de este espacio será función del tipo de variable. La sintaxis de la **declaración de variables** es:

tipo lista_de_nombres;

- **tipo:** indica el *tipo de dato* que se almacena en la variable (entero, flotante, carácter,...).
- **lista_de_nombres:** uno o más nombres de variables separados por comas.

Existen una serie de normas para los **nombres de variables**: pueden contener el carácter guion bajo (_), dígitos numéricos y letras del código ASCII estándar (no pueden contener eñes ni caracteres con tildes). No pueden comenzar por un dígito numérico. Se diferencian mayúsculas y minúsculas.

Ejemplos:

- Nombres correctos: importe3, hora_comienzo, dia, Dia, DIA
- Nombres incorrectos: día, 3importe, año

Tipos básicos de variables:

Nomenclatura	Tipo	Almacenamiento	Rango de valores
int	Entero 32 bits	Complemento a 2	$[-2^{31}, 2^{31}-1]$
unsigned int	Natural de 32 bits	Binario natural	$[0, 2^{32}-1]$
short	Entero de 16 bits	Complemento a 2	$[-2^{15}, 2^{15}-1]$
unsigned short	Natural de 16 bits	Binario natural	$[0, 2^{16}-1]$
long	Entero de 64 bits	Complemento a 2	$[-2^{63}, 2^{63}-1]$
unsigned long	Natural de 64 bits	Binario Natural	$[0, 2^{64}-1]$
float	Real 32 bits	Punto Flotante de simple precisión	$[1.18^{-38}, 3.40^{38}]$ (precisión: 7 dígitos)
double	Real 64 bits	Punto Flotante de doble precisión	$[2.23^{-308}, 1.79^{308}]$ (precisión: 15 dígitos)
long double	Real 80 bits	Punto Flotante precisión extendida	$[3.37^{-4932}, 1.18^{4932}]$ (precisión: 18 dígitos)
char	Entero de 8 bits (carácter*)	Complemento a 2	$[-128, 127]$
unsigned char	Natural de 8 bits (carácter*)	Binario Natural	$[0, 255]$

*El tipo char suele utilizarse para representar el valor numérico de un carácter de la tabla ASCII, debido al rango de valores que abarca. Esto se verá en la práctica de "Cadenas de caracteres".

Constantes

Una *constante* es un dato con un *nombre*, un *tipo* y un *valor* asociado que no puede modificarse una vez definido. La sintaxis de la declaración de una constante es:

#define NOMBRE valor_cte

Las constantes se definen al principio del programa, antes de la función `main()`. No se dice explícitamente de qué tipo es la constante ya que el compilador lo reconoce por el aspecto de *valor_cte*.

Instrucción de asignación

Instrucción que sirve para almacenar un dato en una variable. Todas las sentencias/instrucciones de asignación acaban con el carácter (;).

La sintaxis de la instrucción de asignación es:

Nombre_de_variable = expresión ;

- **nombre_de_variable:** Nombre de variable declarada previamente
- **operador de asignación (=):** Indica que el valor calculado en la expresión debe ser almacenado en Nombre_de_variable.
- **expresión:** Indica cómo se calcula el valor a almacenar en la variable.

Nótese que el sentido de la asignación es de derecha a izquierda

Ejemplo de programa con los conceptos explicados:

```
#define PI 3.14
int main()
{
    double perimetro;
    double radio = 3; /* Declaración de variable con inicialización*/
    perimetro = 2 * PI * radio;
    area = PI * radio * radio ; /* Fallo porque la variable área no está declarada*/
    return 0;
}
```

Expresiones aritméticas

Como ya hemos visto en el epígrafe anterior, podemos asignar valores a las variables mediante una sentencia de asignación. La asignación en lenguaje C tiene la siguiente sintaxis:

variable = expresión;

Tanto las variables como las operaciones de suma que pusimos en el lado derecho de la asignación constituyen dos tipos de expresiones aritméticas básicas del lenguaje C. El resto de expresiones que podemos utilizar las mostramos en el siguiente cuadro:

EXPRESIÓN ARITMÉTICA	Resultado de la expresión
<i>Constante</i>	El resultado de la expresión es el valor de la constante
<i>Nombre_de_Variable</i>	El resultado de la expresión es el valor de la variable
<i>Expresión1 + Expresión2</i>	Suma <i>Expresión1</i> y <i>Expresión2</i> , que son, a su vez, expresiones más pequeñas
<i>Expresión1 - Expresión2</i>	Resta <i>Expresión2</i> a <i>Expresión1</i>
<i>Expresión1 * Expresión2</i>	Multiplíca <i>Expresión1</i> por <i>Expresión2</i>
<i>Expresión1 / Expresión2</i>	Divide <i>Expresión1</i> entre <i>Expresión2</i>
<i>- Expresión</i>	El resultado es la <i>Expresión</i> cambiada de signo (operador unario)
<i>Expresión1 % Expresión2</i>	Operador módulo : Devuelve el resto de la división entera <i>Expresión1</i> entre <i>Expresión2</i>
<i>variable --</i>	Auto-Decremento : Devuelve el valor de la <i>variable</i> ; después resta 1 al valor de la <i>variable</i>
<i>variable ++</i>	Auto-Incremento : Devuelve el valor de la <i>variable</i> ; después suma 1 al valor de la <i>variable</i> .

Veamos algunos ejemplos de expresiones incluyendo algunos errores. Vamos a suponer que en alguna parte de nuestro programa hemos definido la constante *PI* y las variables *radio* y *perímetro*:

```
#define PI 3.141592
double radio = 3000000;
double perimetro;
```

EXPRESIÓN ARITMÉTICA	EJEMPLOS		
<i>Constante</i>	3.32	PI	'A'
<i>Nombre_de_Variable</i>	perimetro	area	Radio
<i>Expresión1 + Expresión2</i>	area+perimetro	3+radio+34	'A'+3
<i>Expresión1 - Expresión2</i>	perimetro-PI	'A'-'a'+3	3+4-5.6-3
<i>Expresión1 * Expresión2</i>	2*PI*radio	PI*radio*radio	PI*(radio+2)
<i>Expresión1 / Expresión2</i>	perimetro/PI	3/2 resultado:1	3.0/2.0 resultado: 1.5
<i>- Expresión</i>	-2	-PI	-sqrt(radio)
<i>Expresión1 % Expresión2</i>	3%2 resultado:1	7%4 resultado:3	7.0%5 ERROR
<i>variable --</i>	radio--	(area+3)-- ERROR	PI-- ERROR
<i>variable ++</i>	area++	sqrt(34)++ ERROR	PI++ ERROR

Como vemos en las tablas, una expresión puede contener como operando otras expresiones, de esta forma podemos escribir algo como:

$$a = 12*x*x + 4*x + 2;$$

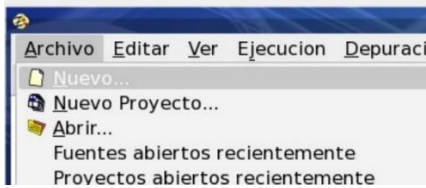

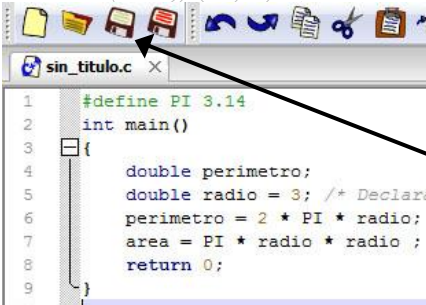
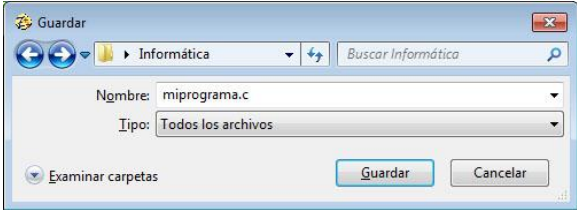
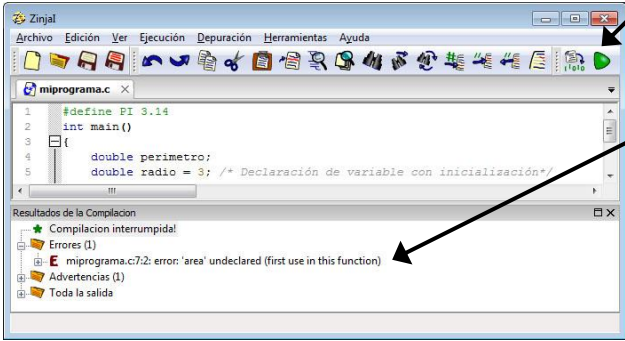
La forma en la que se evalúan o interpretan las expresiones siguen unas reglas similares a las de la aritmética de las matemáticas. Primero se evalúan las de mayor *precedencia* y al final la de menor. La precedencia de las operaciones aritméticas es:

Mayor		Menor	
++, --	- unario	*, / y %	+, -

En el caso de que haya varios operadores de la misma precedencia, se evalúan de izquierda a derecha. Con los paréntesis podemos cambiar la precedencia.

$$a = 12*x*(x + 4)*x + 2; //El paréntesis se evalúa primero$$

PRIMERA PARTE: Creación, edición, compilación y ejecución de un programa C

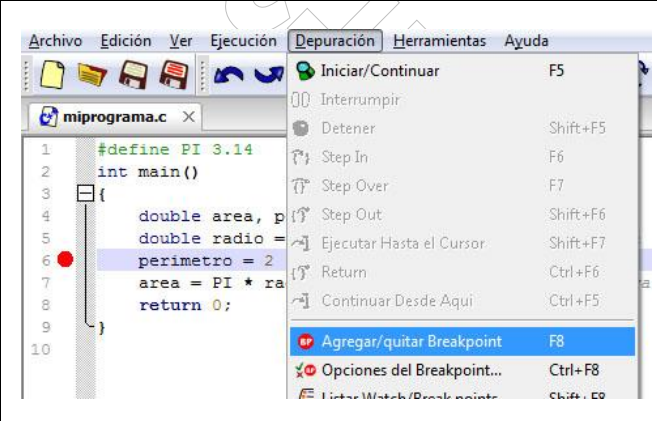

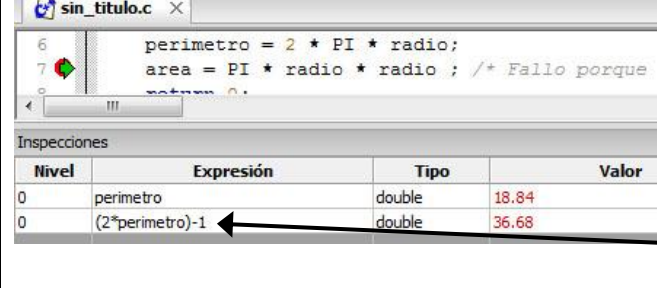
	Para crear un nuevo fichero de código C, haga clic en Archivo→Nuevo o pulse Ctrl+N,
	Seleccione 'Archivo en Blanco'
	<p>En la ventana principal del editor, escriba el programa del ejemplo anterior</p> <p>Salve el programa, pulsando este botón que muestra un disquete, o bien, Ctrl+S en el teclado.</p>
	Elija un lugar del disco duro donde guardar el programa y especifique el nombre del programa más la extensión .c , por ejemplo miprograma.c
	<p>Compile y ejecute el programa pulsando el botón que muestra el play de color verde.</p> <p>Si el programa tuviese errores, el proceso de compilación se interrumpirá y la ventana 'Resultados de compilación' mostrará los errores (y advertencias o warnings) que se han producido. En el ejemplo, se obtiene un error porque la variable area no está declarada.</p> <p>Corrija el error en el programa y pulse play nuevamente para volver a iniciar el proceso de compilación y ejecución.</p>

SEGUNDA PARTE: Depuración de programas

Como se acaba de ver en el apartado anterior, durante el desarrollo del programa es habitual que se cometan errores léxicos, sintácticos y/o semánticos los cuales son notificados por el entorno de desarrollo durante el proceso de compilación en la ventana 'Resultados de Compilación'. Además de estos errores, es posible que se cometan otros relacionados con el comportamiento del programa, esto es, que éste no realice exactamente lo que el programador esperaba.









Una forma de solucionar este problema consiste en depurar el programa realizando una traza al programa, esto es, analizar instrucción tras instrucción la ejecución del programa hasta descubrir qué instrucciones no se ajustan al resultado que se espera. Dicho análisis consistirá fundamentalmente en verificar el valor que va adquiriendo las variables al ejecutar cada instrucción, el flujo del programa, los mensajes de la consola y las llamadas a las funciones. Los entornos de desarrollo como Zinjal suelen integrar una herramienta de depuración para la resolución de problemas de este tipo.

Antes de iniciar la depuración, es necesario que el programa no muestre errores de compilación en la ventana de 'Resultados de la Compilación'. Para depurar el programa anterior, haga lo siguiente:

	<p>Primeramente deberá establecer los puntos de ruptura (Breakpoints). Estos puntos de ruptura permiten especificar en qué instrucciones se desea detener la ejecución con el fin de poder iniciar la depuración desde tales instrucciones. Para añadir un punto de ruptura sitúe el cursor en la instrucción deseada y seleccione en el 'Depuración → Agregar/quitar Breakpoint', o bien, pulse F8. Podrá identificar qué instrucciones tienen puntos de ruptura mediante el círculo rojo que aparece junto a la instrucción.</p>												
	<p>Inicie la depuración del programa desde el menú 'Depuración → Iniciar/Continuar', o bien, pulsando F5. Al iniciar la depuración la ejecución del programa comenzará y se detendrá en el primer punto de ruptura que encuentre. Una vez detenida la ejecución, podrá identificar cual es la siguiente instrucción que se va a ejecutar mediante la flecha verde que aparecerá junto a la instrucción.</p>												
 <table border="1"><thead><tr><th>Nivel</th><th>Expresión</th><th>Tipo</th><th>Valor</th></tr></thead><tbody><tr><td>0</td><td>perimetro</td><td>double</td><td>18.84</td></tr><tr><td>0</td><td>(2*perimetro)-1</td><td>double</td><td>36.68</td></tr></tbody></table>	Nivel	Expresión	Tipo	Valor	0	perimetro	double	18.84	0	(2*perimetro)-1	double	36.68	<p>Desde el menú 'Depuración' puede abrir todas las ventanas relacionadas con la depuración. Una de las más interesantes es 'Panel de Inspecciones' pues muestra el valor de las variables durante la ejecución. Para añadir una variable deberá escribirla en el campo 'Expresión' de una línea vacía. Además de variables puede añadir incluso expresiones que contengan dichas variables.</p>
Nivel	Expresión	Tipo	Valor										
0	perimetro	double	18.84										
0	(2*perimetro)-1	double	36.68										

Al detenerse la ejecución y entrar modo depuración, aparecerán nuevos botones que permitirán controlar la depuración.

Las opciones que ofrece el depurador son las siguientes:

	Iniciar/Continuar: Ejecuta una o varias instrucciones hasta el siguiente punto de ruptura o el fin del programa.
	Detener (Shift + F5): Finaliza la depuración y vuelve al editor.
	Step In (F6): Ejecuta la siguiente instrucción. Si ésta fuese una llamada a función muestra como próxima instrucción a ejecutar la primera instrucción de la función a la que se ha llamado. Esta opción se utiliza por tanto cuando se desea depurar también la función a la que se llama.
	Step over (F7): Ejecuta la siguiente instrucción. Si fuese una llamada a función no se depura el interior de la función por lo que se muestra como próxima instrucción a depurar la que le sucede a la llamada. Por tanto, esta opción trata a las funciones como instrucciones simples y debe usarse para evitar “entrar” en funciones que ya sabemos que funcionan correctamente.
	Step out (Shift + F6): Ejecuta una o instrucciones hasta finalizar la función en la que se encuentra detenido el depurador. Puede usarse cuando, por algún motivo, no se desea continuar depurando la función actualmente en depuración.
	Ejecutar hasta el cursor: Ejecuta una o varias instrucciones hasta alcanzar la instrucción en la que se encuentra el cursor. Resultar útil para reanudar la depuración a partir de otro punto del programa.
	Return: interrumpe la ejecución de la función actual y emplea como valor de retorno de la función el que el programador introduzca por teclado.
	Continuar desde aquí: salta la ejecución del programa al punto en el que se encuentra el cursor. Esta opción no ejecuta instrucción alguna, simplemente establece la próxima instrucción por la que continuar la depuración.

Puede consultar documentación más detallada sobre este entorno de desarrollo en la web oficial:

<http://zinjai.sourceforge.net/index.php?page=documentacion.php>

Si se ha descargado para Windows la versión completa (full), la ayuda del entorno está disponible dentro del mismo.

EJERCICIOS

1. Crear al menos una variable de cada uno de los tipos explicados, asignarles diversos valores y comprobar mediante el depurador la asignación de los mismos.
2. Crear una constante llamada *PI* cuyo valor sea el del n° pi. Asignar el valor de *PI* a las variables de distinto tipo creadas en el punto anterior.
3. Intentar asignar un valor a *PI* dentro de nuestra función *main*. ¿Es posible?
4. Escribir en un nuevo programa el siguiente código:

```
#include <stdio.h>
int main()
{
    int a, b;

    a = 10;
    b = 20;
    b = a;
    a = 40;
    return 0;
}
```

- ¿Cuál es el valor de *a* en la primera asignación? ¿Y el de *b* tras la primera asignación?
- ¿Cambia el valor de *b* tras asignarle un nuevo valor a la variable *a* (*a=40*)?

5. Indique los errores, si los hay, en las siguientes líneas de código:

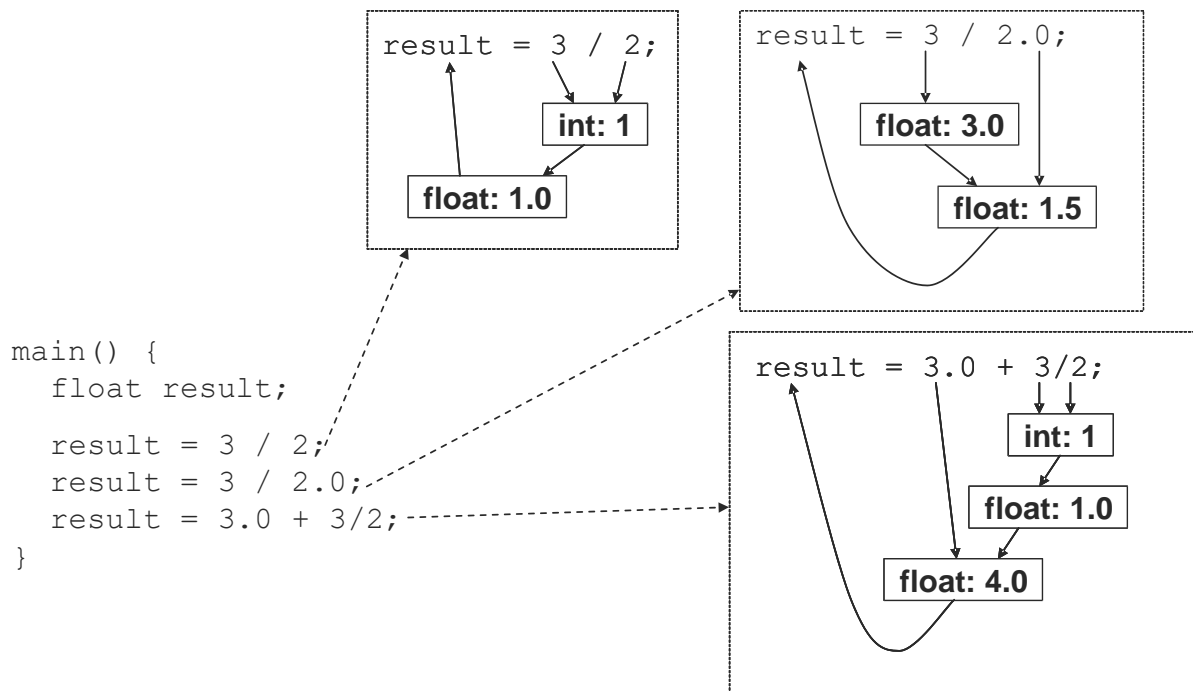
- `int a = int b = 0;`
- `a + b = c;`
- `int decimal = 4.99;`
- `char letra = 51;`
- `int numero = 'a';`
- `char frase = 'hola, mundo';`

6. Dado el siguiente código:

```
int main() {  
    float result;  
    result = 3 / 2;  
    result = 3 / 2.0;  
    result = 3.0 + 3/2;  
}
```

Realizar las operaciones en papel y comprobar el valor calculado por el ordenador mediante la depuración. ¿Por qué no realiza la operación como esperábamos?

Solución: Conversión implícita de datos



7. Escriba un programa partiendo del siguiente código que se le proporciona, donde se calcule el perímetro y el área de una circunferencia:

```
#define PI 3.14

int main()
{
    float radio = 4.2;
    float perimetro;
    float area;
}
```

Pruebe a cambiar el valor de la variable radio y compruebe si el resultado es correcto por medio del depurador.

8. Dado el siguiente código:

```
#define CONST=19;
mains() {
    int a = 1 b c;
    a + 2 = b;
    3 = 4 * c;
    resultado = c - (b + a)
}
```

Identifique los errores que presenta y corríjalos como crea oportuno. Luego, utilice el depurador para ver el resultado obtenido almacenado en la variable “resultado”.

9. *Análisis de código.* Analice el siguiente fragmento de código **SIN utilizar ZinjaI** y escriba en la tabla los valores que vayan tomando cada una de las variables:

```
int main()
{
    int a;
    float b, c, resultado;

    a = 5.5;
    b = 2 + 1/2;
    a = a + b;
    c = 1/2.0;
    c = c + 1;

    resultado = (a+c)*b;
}
```

a	b	c

¿Qué valor toma finalmente la variable “resultado”?




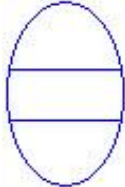
DIAGRAMA DE FLUJOS

El diagrama de flujos es una herramienta empleada para representar gráficamente un proceso o algoritmo, es por ello que son utilizados en programación para describir el comportamiento de un programa. Estos diagramas serán de utilidad para el alumno para definir cómo se va a desarrollar el programa antes de programarlo en lenguaje C. Puesto que el entorno de desarrollo ZinjaI es capaz de representar el diagrama de flujo, el alumno podrá utilizar también los diagramas para para verificar si el programa que ha realizado se ajusta a sus especificaciones.

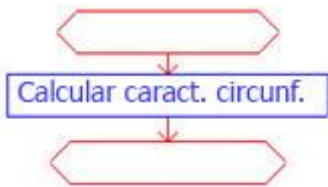
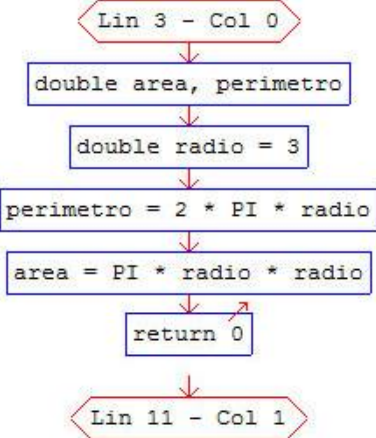
Los diagramas de flujos se basan fundamentalmente en la utilización de:

- **SÍMBOLOS o CAJAS:** donde se escriben las acciones que realiza el algoritmo.
- **LÍNEAS DE FLUJO o FLECHAS:** conectan los símbolos e indican la secuencia en la que se ejecutan las acciones.

Aunque los diagramas recogen muy diversos tipos de símbolos, en esta asignatura sólo se tratarán principalmente los siguientes:

Símbolo	Nombre	Descripción
	Terminador	Marca el <i>inicio</i> y <i>fin</i> del diagrama de flujo, por tanto especificarán el punto en el que empieza y termina la ejecución del programa (o función).
	Proceso	Indica la tarea a realizar, la cual se encuentra descrita en su interior.
	Condición	Selecciona un flujo u otro en función de si se cumple (v) o no (f) la pregunta/condición que se especifica en su interior Se utiliza para representar <i>estructuras selectivas</i> y <i>iterativas</i> y que se tratarán en posteriores prácticas.
	Iteración	Repite varias tareas mientras se cumple una condición. Representa a <i>estructuras iterativas</i> con <i>for</i> y se detallará su comportamiento en posteriores prácticas.

El diagrama de flujo del programa anterior sería:

Realizado por el programador	Generado por ZinjaI
	

Como puede observarse, el programador indicará en cada símbolo la tarea que se va a realizar sin necesidad de especificar qué instrucción o instrucciones en C son las que realizan la tarea ya que expresa el comportamiento del programa en un lenguaje más natural (de más alto nivel). En cambio el generador de ZinjaI simplemente crea un símbolo por cada instrucción o declaración ya que genera el diagrama de flujo ajustándose al código que ya posee. En este caso, dada la simplicidad del programa empleado como ejemplo, no justifica la necesidad de diseñar un diagrama de flujo. En posteriores prácticas, el uso de diagramas de flujo puede ayudar a describir y plantear cómo va a desarrollar el programa.