Universidad de Santiago de Chile
Departamento de Ingeniería Informática
Métodos de Programación
Guía de preparatoria para las actividades de laboratorio
Documento N°1

# 1 Introducción

Este es el primer documento que les servirá a usted como apoyo a la materia que se estará realizando en el curso.

En esta oportunidad se hace una introducción al lenguaje de programación C, el cual servirá de apoyo para la implementación de los métodos de resolución de problemas que se enseñarán principalmente en cátedra.

En el documento se desarrolla el contenido de algunos conceptos, como lo son paradigma y lenguaje de programación, después se introduce al lenguaje de programación C, indicando operadores básicos y tipos de datos, para finalizar esta parte con lo que es el compilar y ejecutar. Posteriormente se desarrolla un ejercicio con el fin de explicar la materia, para dejar, finalmente, unos ejercicios que le servirán a usted avanzar con esta materia.

# 2 Desarrollo

## 2.1 Paradigma de programación

Antes de entender que es un paradigma de programación es necesario saber que es un paradigma por sí mismo.

Este concepto, es muy utilizado, pero normalmente poco entendido, se puede decir que un paradigma es algo que se toma como ejemplo para realizar cierta cosa. Inicialmente este era considerado un patrón o modelo acorde a su etimología, y existen muchas personas que lo han definido acorde a su área, como Thomas S. Kuhn, orientando una definición más científica, o Fritjof Carpa, quien da un concepto más amplio, relacionando la historia, civilizaciones y la naturaleza.

Los paradigmas de programación es una aproximación a la programación basada en conceptos matemáticos o en la coherencia de un conjunto de principios. Dentro de los paradigmas de programación podemos encontrar la programación Orientada a objetos (POO) que es muy útil utilizar cuando existe información organizada que tiene interacción entre sí, o tenemos también la programación lógica, que es mejor para ir de un lado a otro en problemas complejos que poseen reglas lógicas.

En el curso anterior, Fundamentos de Computación y Programación, usted debe haber trabajado en el lenguaje de programación Python, donde le indicaron que es un lenguaje multiparadigma, es decir, puede utilizar más de un principio para resolver un problema, pero que en el curso se trabajaría con el paradigma imperativo, mismo paradigma que es utilizado en este curso, pero aplicando otro lenguaje de programación.

## 2.2 Lenguaje de programación

Antes de hablar del lenguaje de programación, se debe tener en cuenta que en todos los lenguajes en sí, los que inicialmente poseen tres características o propiedades, las cuales se denominan:

- **Reglas léxicas:** Corresponde a las palabras que existen dentro del lenguaje, y no solo palabras, hay que recordar que existen lenguajes compuestos de símbolos o señas.
- **Reglas sintácticas:** Corresponden a aquellas que indican cómo las palabras se deben mezclar las palabras o símbolos correspondientes, por ejemplo, cuando se nos enseña a construir oraciones en el español, una regla sintáctica es que las palabras del grupo Sujeto deben ser seguida por una palabra del conjunto de los verbos, para finalizar todo con un conjunto de palabras correspondientes al complemento.
- **Reglas semánticas:** Corresponde a aquellas reglas que ayudan a que las frases formadas en el lenguaje posean un sentido.

Entonces, mediante esto, y tomando como objetivo que los lenguajes de programación son necesarios para dar instrucciones a un computador, se puede definir que un lenguaje de programación es un conjunto de reglas léxicas, sintácticas y semánticas que permiten al ser humano comunicarse con las computadoras.

Dentro de los lenguajes de programación se puede encontrar una división, la de lenguajes de alto nivel, correspondiente a lenguajes que se asemejan mucho al lenguaje humano; por ejemplo, Python. Y los lenguajes de bajo nivel, que se asemejan mucho más a los lenguajes de máquina, por ejemplo, assembly o ASM.

## 2.3 Lenguaje C

El lenguaje de bajo nivel, desarrollado por Dennis Ritchie entre los años 1969 y 1972, en conjunto al Sistema Operativo (S.O.) Unix, como propuesta al lenguaje B[1] y es un lenguaje altamente tipado, es decir, si una variable es declarada de un tipo de dato en específico, esta no puede modificar ese tipo.

### 2.3.1 Tipo de Datos

En el lenguaje C, los tipos de datos que existen son los siguientes:

- **Enteros:** Corresponde al tipo de dato más primitivo de C, utilizado principalmente para representar números enteros, pero se puede utilizar para representar cualquier tipo de dato del tipo discreto. Dentro de los tipos de datos enteros están los *short*, *int*, *long* y *long long*, junto con su distinción con signo (*signed*) o sin signo (*unsigned*).
- **Reales:** Se representan los números reales utilizando la técnica de coma flotante[2], el cual ayuda a la operación de números con distintas magnitudes, separando el decimal (mantisa) del exponente (orden de la magnitud). Los tipos de datos reales utilizado en este tipo de representación son los *float* y *double*, que poseen la diferencia de la cantidad de bytes que utilizan.

[1] Abreviación del lenguaje de nombre *Basic Combined Programming Language* (BCPL)

[2] Puede encontrar información sobre esta técnica en la siguiente página web: *https://es.wikipedia.org/wiki/Coma_flotante*

- **Caracteres:** Los char en C son un tipo de dato que posee solo un byte de tamaño, los cuales pueden representar hasta 256 elementos (cada uno corresponde a un valor de la tabla de caracteres del sistema (código ASCII)). Este tipo de dato puede ser *signed* o *unsigned*.

En la tabla 2.1 se muestran los distintos tipos de datos primitivos del lenguaje C y sus principales características.

Tabla 2.1: Tipos de datos primitivos de C.

| Tipo | Representa | Tamaño en la RAM | Rango de Representación (mínimo-máximo) |
|---|---|---|---|
| **char** | Caracteres | 1 byte (8 bits) | 0 a 255 |
| **int** | Números Enteros | 4 bytes (*) (32 bits) | -2.147.483.648 a 2.147.483.647 |
| **float** | Números en punto flotante | 4 bytes (**) (32 bits) | (±) 3.4E-38 a 3.4E38 |
| **double** | Números en punto flotante de doble precisión | 8 bytes (***) (64 bits) | (±) 1.7E-308 a 1.7E308 |

### 2.3.2 Operadores básicos

Dentro de la programación, no solo los tipos de datos son importantes, sino que también dentro de esto están los operadores. En este punto, el principal operador que existe es el llamado de asignación, encontrando también otros como aritméticos y los lógicos.

En la tabla 2.2 se muestran los operadores que más se utilizarán dentro del curso, pero puede encontrar más información en: *http://lsi.vc.ehu.es/asignaturas/FdIc/labs/a1/htm/oper.html*.

Tabla 2.2: Operadores en C.

| Tipo | Nombre | Símbolo | Descripción |
|---|---|---|---|
| **Asignación** | Asignación | = | Operador de asignación. Además de este operador de asignación, existen otros que realizan la acción, siendo operadores de asignación de operaciones más complejas, como lo son los operadores +=, -=, *= y otros. Además, están los operadores de incremento pre y post ++ y --. |

| | | | |
|---|---|---|---|
| **Aritmético** | Suma | + | Suma o adición |
| | Resta | - | Resta o sustracción |
| | Multiplicación | * | Multiplicación o producto |
| | División | / | División, cociente de la división entera |
| | Módulo | % | Módulo o resto de la división entera |
| **Relacionales** | Mayor | >, >= | Mayor y mayor o igual |
| | Menor | <, <= | Menor y menor o igual |
| | Igualdad | == | Igual |
| | Distinto | != | Distinto |
| **Lógicos** | Conjunción | && | Conjunción o Y lógico |
| | Disyunción | \|\| | Disyunción u O lógico |
| | Negación | - | Negación o NO lógico |

La resolución de las operaciones aritméticas y lógicas dentro del lenguaje C, son similares a las vistas en C, por lo que existe una estructura jerarquizada la cual instrucción se realiza antes que otra, lo cual se llama prioridad de operaciones. Esta prioridad de los operadores se muestra en la Tabla 2.3.

Tabla 2.3: Precedencia de operadores[3]

| Nivel | Operadores | Descripción | Asoci. |
|---|---|---|---|
| 1 | `() [] -> .` | Acceso a un elemento de un vector y paréntesis | Izquierdas |
| 2 | `+ - ! ~`<br>`* &`<br>`++ --`<br>`(cast) sizeof` | Signo (unario), negación lógica, negación bit a bit<br>Acceso a un elemento (unarios): puntero y dirección<br>Incremento y decremento (pre y post)<br>Conversión de tipo (*casting*) y tamaño de un elemento | **Derechas** |
| 3 | `* / %` | Producto, división, módulo (resto) | Izquierdas |
| 4 | `+ -` | Suma y resta | Izquierdas |
| 5 | `>> <<` | Desplazamientos | Izquierdas |
| 6 | `< <= >= >` | Comparaciones de superioridad e inferioridad | Izquierdas |
| 7 | `== !=` | Comparaciones de igualdad | Izquierdas |
| 8 | `&` | Y (*And*) bit a bit (binario) | Izquierdas |
| 9 | `^` | O-exclusivo (*Exclusive-Or*) (binario) | Izquierdas |
| 10 | `\|` | O (*Or*) bit a bit (binario) | Izquierdas |
| 11 | `&&` | Y (*And*) lógico | Izquierdas |
| 12 | `\|\|` | O (*Or*) lógico | Izquierdas |
| 13 | `?:` | Condicional | **Derechas** |
| 14 | `= *= /= %= += -= >>= <<= &= ^= \|=` | Asignaciones | **Derechas** |
| 15 | `,` | Coma | Izquierdas |

## 2.4 Compilar v/s interpretar

En el mundo de la programación existen dos tipos de lenguajes, aquellos interpretados, como los que usted debió haber visto en el curso de Fundamentos de computación y programación, y aquellos que son compilados, como es el caso del lenguaje C.

Para explicar ambos casos, se define inicialmente el concepto de código fuente, el cual corresponde al código (programa, texto, archivo u otros nombres) que posee las instrucciones que en un lenguaje de programación se le está dando al computador para que realice lo que el programador quiere, este siempre está en un lenguaje de programación conocido por el programador. Por otro lado, se tiene el lenguaje máquina corresponde al lenguaje con que la máquina entiende directamente.

Los intérpretes, realizan la ejecución directa línea a línea del código fuente, realizando los pasos de leer la instrucción en el lenguaje de programación dado, esa instrucción la traduce al lenguaje máquina, y posteriormente ésta lo ejecuta. Podemos tomar como ejemplo de esto a un intérprete humano en una entrevista, primero habla el entrevistado, el cual da una frase y se detiene para que luego el intérprete, diga lo mismo en el lenguaje deseado, posteriormente el

[3] Tabla obtenida de: http://lsi.vc.ehu.es/asignaturas/FdIc/labs/a1/htm/oper.html

entrevistado continúa con otra frase, y el proceso se repite hasta que el intérprete haya dicho todo lo que el entrevistado haya dicho.

Por el otro lado está el compilador, quien, a diferencia del intérprete, este traduce todo el código fuente en lenguaje máquina, no de línea en línea, para posteriormente crear un código intermedio, al cual llamaremos código ejecutable, y ese resultado será el que la máquina deberá realizar. Se puede hacer un símil a lo que un traductor realiza con un libro, recibiendo el libro completo, para luego pasarlo a un lenguaje o idioma objetivo.

Las diferencias de compilador e intérprete se muestran en la tabla 2.4.

Tabla 2.4: Algunas diferencias entre intérprete y compilador.

| Intérprete | Compilador |
|---|---|
| Realiza la "traducción" en el momento que se ejecutará la instrucción. | Realiza la "traducción" a todo el código fuente de una sola vez. |
| No genera un archivo nuevo con código intermedio o código máquina. | Genera un archivo nuevo con código intermedio o código máquina llamado código ejecutable o simplemente ejecutable. |
| Si hay cambios en las instrucciones del programa, se puede seguir con el mismo procedimiento de interpretar el código. | Si hay cambios en las instrucciones necesario que el código se deba compilar completamente. |
| Al interpretarse, cada vez se ejecuta el último código fuente, con los últimos cambios. | Si no se compila nuevamente, y se hacen cambios en el código fuente, se estará ejecutando código antiguo. |
| Si el código se mantiene, de todas formas se debe realizar el mismo procedimiento de interpretación, como si fuera un código fuente nuevo | Si el código se mantiene, no es necesario volver a compilar, se debe ejecutar solo el último ejecutable creado. |

## 2.5 Compilar y ejecutar código en C

Para trabajar en el curso, se utilizarán dos herramientas, la primera será un editor de texto plano, que puede el mismo editor de texto que trae Windows o los sistemas operativos UNIX, o puede ser algún editor más sofisticado como Notepad++[4] o Sublime Text[5]. Por otro lado, se utilizará el compilador GCC (*GNU Compiler Collection*) que en la mayoría de los sistemas operativos UNIX o Mac viene por defecto en la terminal, o se debe instalar mediante el uso de la Shell con instrucciones como:

- sudo **<gestorRepositorio>** install gcc

[4] Páguina oficial: *https://notepad-plus-plus.org/*
[5] Páguina oficial: *https://www.sublimetext.com/3*

- sudo **apt-get** install gcc (Para sistemas como Debian o Ubuntu)

En cuanto a sistemas operativos Windows se deberá instalar el programa MinGW y posteriormente agregar los respectivos PATH a las variables de entorno del sistema[6].

Ya teniendo esto, es posible escribir nuestro primer código, el cual se puede ver en la Figura 2.1, la cual es explicada línea a línea en la tabla 2.5.

Figura 2.1: Hola mundo en C.

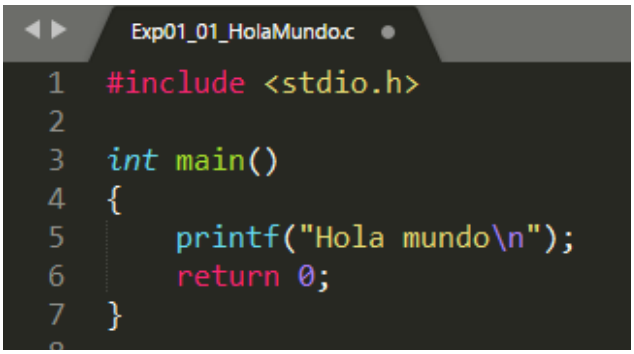



Tabla 2.5: Explicación del código, línea a línea

| Línea | Instrucción/ Comando | Explicación |
|---|---|---|
| 1 | #include | Es una palabra reservada del lenguaje[7] que sirve para "incluir" otro fichero dentro del programa. Para más detalle puede ver en el siguiente link: *https://trucosinformaticos.wordpress.com/2012/10/14/como-usar-include-en-c-y-c/* |
| 1 | stdio.h | Librería estándar de C para temas de entradas y salidas. En este caso se utiliza para la función *printf*. Puede encontrar más información de esta librería y otras útiles en el lenguaje en el siguiente link: *https://www.tutorialspoint.com/c_standard_library/stdio_h.htm* |
| 3 | int | Valor de retorno de la función, en este caso del main, Este tipo de dato debe coincidir con todos los retornos que posea la función, en este caso la línea 6, que retorna un valor entero, en este caso, un 0. |
| 3 | main | Es el nombre de la función principal del código, es aquella que al encontrarse, cuando se lee el código desde arriba a abajo, se ejecuta. Se podría decir que es |

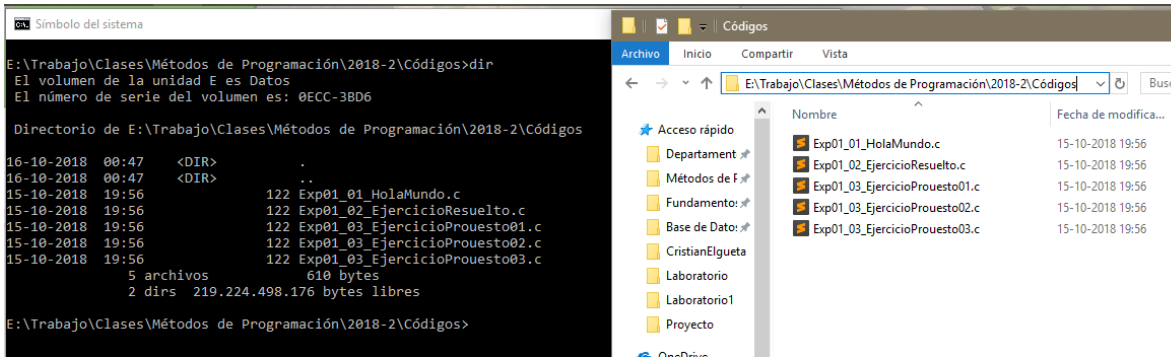
[6] En este link puede ver como realizar el proceso completo de instalación de MinGW: *https://www.fdi.ucm.es/profesor/luis/fp/devtools/MinGW.html*

[7] Observación: es toda la estructura de la palabra, es decir, include no es reservada, pero #include sí lo es.

| | | |
|---|---|---|
| | | como el símil con el bloque principal que ustedes hacían en sus programas en Python.<br>Esta función, al igual que todas las otras que se declaran en este lenguaje, después del nombre llevan los argumentos de entrada, que en este caso está vacío razón del "()", posteriormente, el cuerpo de la función está declarado entre las llaves "{" y "}". Es necesario indicar, que a diferencia de Python, acá la identación no es obligatoria, pero sí es útil y necesaria como buena práctica de programación. |
| 5 | printf | Función importada desde la librería stdio.h, sirve para enviar un mensaje por la entrada estándar.<br>En este caso solo se le está pasando el texto que se enviará por pantalla, pero también se pueden enviar valores por pantalla, tema que se mostrará en la sección 3.<br>Todas las instrucciones deben finalizar con un ";". |
| 6 | return | Instrucción que hace que finalice lo que está realizando la función, y retorna el valor que viene posteriormente. En este caso en específico, como es el return de la función main, hará que finalice el programa en sí.<br>El valor de retorno debe ser del mismo tipo de dato que se declara en el retorno de la función, en este caso, la línea 3. |

Ya teniendo el código, debemos abrir la terminal (buscar los sistemas operativos UNIX escribiendo *terminal*, o en Windows *cmd*), en la Figura 2.2 se puede ver que la consola y los archivos se encuentran en la misma ubicación.

Figura 2.2: Comparación de ubicación y archivos en consola y visualización de Windows.



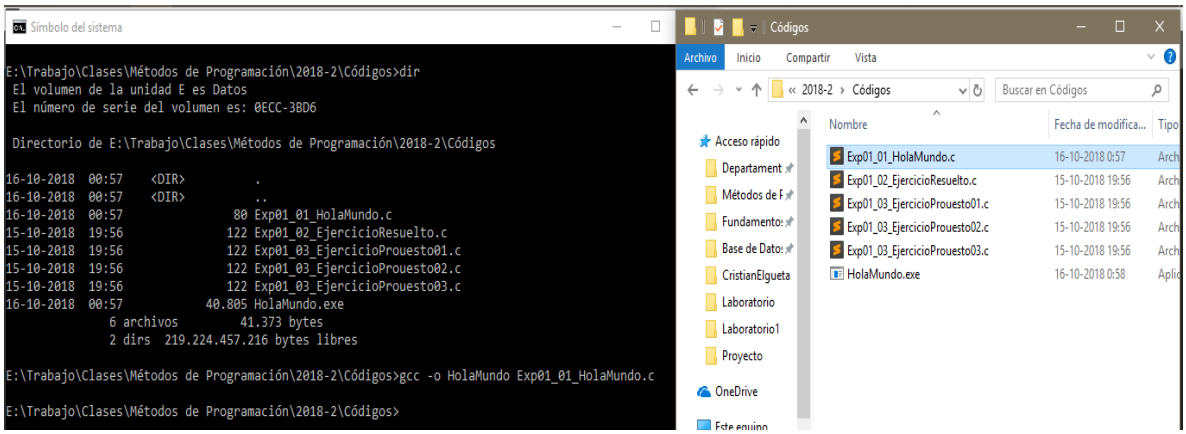

Posteriormente, en la consola se escribe el comando necesario para compilar, de la siguiente forma:

- gcc -o **<nombre del archivo de Salida> <nombre del código fuente>**[8]

[8] También existe otra forma, lo cual no produce diferencia:

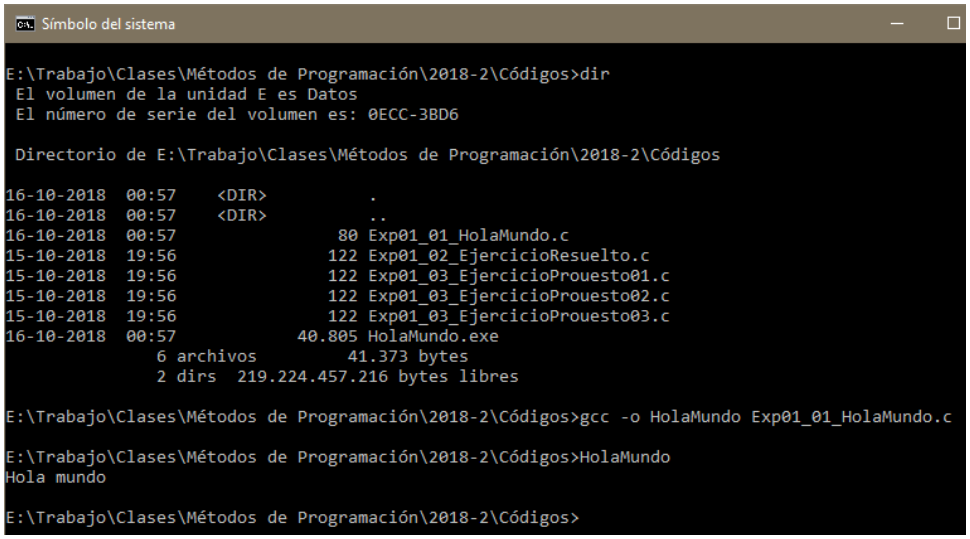
Con esto se generará un nuevo archivo, en el caso de windows será uno con extensión .exe (ver la Figura 2.3)

Figura 2.3: Compilación y creación del archivo ejecutable.




Para ejecutar, se debe escribir el nombre del ejecutable, sin la extensión (para los sistemas operativos UNIX, se debe escribir el nombre del ejecutable, precedido por ./), generando la salida “Hola mundo”, tal como se muestra en la Figura 2.4.

Figura 2.4: Respuesta de la ejecución.



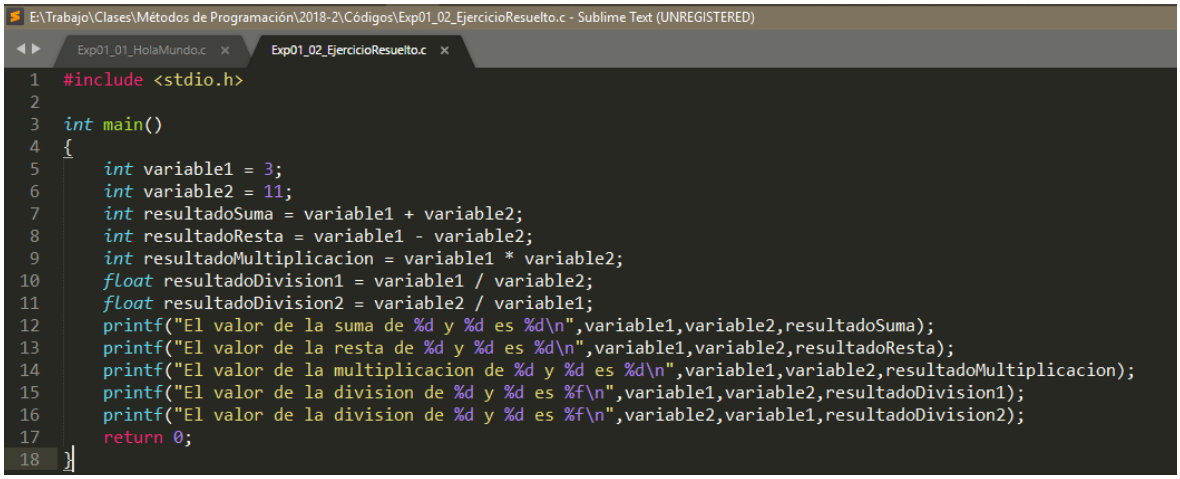

gcc **<nombre del código fuente>** -o **<nombre del archivo de Salida>**

# 3 Ejercicio resuelto

## 3.1 Contexto

El siguiente ejercicio posee la idea de mostrar qué es lo que sucede con algunos elementos que no se han visto dentro del documento en sí, teniendo en cuenta algunos detalles. En la Figura 3.1 muestra el código a analizar.

Figura 3.1: Código de análisis como ejercicio resuelto.




## 3.2 Explicación de la solución

En el código mostrado de la Figura 3.1 hay unas líneas de código distintas a las ya mencionadas en la Tabla 2.4 o que han sido modificadas.
La primera diferencia son las líneas 5 a la 11, en donde se define una variable, llamada variable local de la función main. Estas líneas de código poseen todas la misma estructura, la cual es la siguiente:

- **<Tipo de dato de la variable> <Nombre de la variable> = <Valor variable>;**

Dónde el **<Tipo de dato de la variable>** corresponde a un tipo de los datos primitivos de C, los cuales fueron vistos en la sección 2.3.1. El nombre de la variable corresponde a cómo la llamaremos durante el resto del programa, la cual debe tener como reglas, las mismas que poseían las variables cuando fueron vistas en Python, es decir, deben estar escritas en estilo camelCase, representativas, no pueden comenzar con números, y otros.

El **<Valor variable>** es un elemento optativo, que indica que se está dando un valor inicial a la variable, o que se dará después (siempre se debe dar un valor inicial, lo cual puede ser en el momento de la definición o más adelante) y el valor debe corresponder al tipo de dato que se está indicando. Como se puede notar, el valor de la variable puede ser un número directamente o resultados de operaciones aritméticas, de la misma forma como lo hicieron en el curso anterior.

Por otro lado están las líneas 12 a la 16, la función printf es un poco distinta, ya que esta posee una estructura distinta, que es más o menos del estilo:

- *printf(**<texto a mostrar>**, **<valor de variable a mostrar>**, …)*

Dónde el **<texto a mostrar>** corresponde al texto que se le quiere mostrar al usuario por pantalla, pero con la acotación que cuando se quiere mostrar el valor de una variable, se debe colocar una bandera o *flag*, la cual posee la estructura de **%<Especificación del tipo de dato>**, en donde se pueden ver dos especificaciones de tipos de datos la **d** (para indicar enteros) y la **f** (para indicar números de punto flotante). En la tabla 3.1 se muestran los distintos valores para especificar la salida, aunque puede ver más especificaciones en:

*https://www.tutorialspoint.com/c_standard_library/c_function_printf.htm*

Tabla 3.1: Especificaciones y salidas de la función *printf*.

| Especificador | Tipo de salida | Especificador | Tipo de salida |
|---|---|---|---|
| c | Carácter (**char**) | d o i | Entero con signo en base decimal (**signed int**) |
| f | Número de punto flotante (**float**) | O | Número entero en base octal (**signed int**) |
| s | Cadena de caracteres (***char**) | u | Entero sin signo (**unsigned int**) |

## 3.3 Solución final

Acorde a lo mostrado en la sección 3.1 y 3.2, la salida del código es la mostrada en la Figura 3.2.

Figura 3.2: Salida del código del ejercicio resuelto.

```
E:\Trabajo\Clases\Métodos de Programación\2018-2\Códigos>operacionesAritmeticas
El valor de la suma de 3 y 11 es 14
El valor de la resta de 3 y 11 es -8
El valor de la multiplicacion de 3 y 11 es 33
El valor de la division de 3 y 11 es 0.000000
El valor de la division de 11 y 3 es 3.000000
```

# 4 Ejercicios propuestos

## 4.1 Ejercicio N°1:

Realice un programa que posea tres variables del tipo entero (**int**), llamadas a, b y c con los valores 3, 12, 21 respectivamente. Muestre por pantalla el resultado de las siguientes expresiones, donde ambos resultados sean almacenados en variables del tipo entero (**int**):[9]

- $\frac{c}{a} + \frac{b}{a}$
- $\frac{a+b+c}{a} + \frac{b-a}{c-b}$

## 4.2 Ejercicio N°2:

Realice un programa que posea tres variables del tipo entero (**int**), llamadas a, b y c con los valores 3, 12, 21 respectivamente. Muestre por pantalla el resultado de las siguientes expresiones, donde ambos resultados sean almacenados en variables del tipo entero (**int**):[10]

- $\frac{c}{a} + \frac{b}{a}$
- $\frac{a+b+c}{a} + \frac{b+a}{c+b}$

Realice el proceso con calculadora, y compruebe los resultados, en caso de ser distintos, realice las conversiones de tipo de datos necesarias para que de el mismo resultado.

## 4.3 Ejercicio N°3:

Realice un programa en dónde defina una variable llamada PI del tipo punto flotante (float) y una llamada radio del tipo entero (int), la cual representará el radio de una circunferencia, muestre por pantalla el perímetro y el área de ésta.

[9] Los resultados son 11 y 13 respectivamente.
[10] Los resultados son 11 y 12 respectivamente, debería ser 11.0 y 12.45.

# 5 Bibliografía


- Carpa, Fritjof (1999). El punto Crucial. Buenos Aires. Editorial Estaciones.
- Dennis M. Ritchie (1993), The Development of the C Language. Estados Unidos de América, *Bell Labs/Lucent Technologies.*
- Kuhn, Thomas.(1962): "*The Structure of Scientific Revolutions"* Estados Unidos de América. *University of Chicago Press*.
- Real Academia Española (RAE). "Paradigma | Definición de Paradigma - Diccionario de la Lengua Española - Edición del Tricentenario. Disponible en http://dle.rae.es/?id=RpXSRZJ (Visitada 05/10/2018)
- Significados: Descubrir lo que significa, conceptos y definiciones, (2013-2018) *"Significado de Lenguaje"* (https://www.significados.com/lenguaje/) (Visitada 05/10/2018)
- Van Roy, Peter (2014). Programming Paradigms for Dummies. What Every Programmer Should Know". (https://www.info.ucl.ac.be/~pvr/VanRoyChapter.pdf) (Visitada 05/10/2018)