lTP Nº 5: Estructuras de control

Objetivos:

1. Incorporar los conceptos de la programación estructurada, que basa la resolución de problemas en la utilización de tres estructuras básicas: secuencia, selección y repetición.
2. Realizar el análisis del problema planteado, previo al desarrollo del algoritmo, aplicando el método de resolución de problemas.
3. Entender y aplicar las distintas estructuras utilizando la más adecuada al problema.
4. Reforzar el concepto de funciones y alcance de las variables.
5. Incorporar la verificación con Casos de Prueba que permitan determinar el correcto funcionamiento del algoritmo, es decir si el resultado obtenido es el resultado deseado.

Resolver los ejercicios, respetando en cada caso, las siguientes consignas:

- Aplicar el método de resolución de problemas: definir el objetivo, pensar una estrategia y descomponerla en partes.
- Codificar la solución en lenguaje C.
- Realizar la verificación con casos de prueba representativos del problema. Tener en cuenta que es necesario identificar de antemano cuáles son los resultados esperados para cada caso de prueba.

1. ESTRUCTURA SECUENCIAL

a) Diseñe un algoritmo que emita el ticket con el importe a pagar en un supermercado considerando los datos: cantidad y precio del producto. Imprimir el ticket tomando como referencia el diseño de este ticket de ejemplo:

```
Cant. unidades: 5 - Precio unit.: $ 10.50
Total: $ 52.50
¡Gracias por su compra!
```

b) Diseñe un algoritmo para calcular y mostrar el porcentaje de partidos de fútbol ganados por un club en un campeonato, conociendo los datos: cantidad de partidos jugados y cantidad de partidos ganados por el club.

2. ESTRUCTURAS DE CONTROL ALTERNATIVA

Al resolver un problema podría pasar que se requiera elegir **uno de entre varios caminos** en función de ciertas condiciones.

Por ejemplo, si tuviera que resolver el problema que un robot salga de una habitación, en donde la puerta puede o no estar abierta, se debería indicar de alguna manera que en el caso que la puerta esté cerrada, primero hay que abrirla.

Para resolver situaciones como esta y otras más, en programación existen las **sentencias alternativas, selectivas o de decisión** que permiten como resultado de la evaluación de una condición, seleccionar uno de entre varios caminos por donde seguirá la ejecución del programa.

2.1. Alternativa Simple

La estructura **alternativa simple** ejecuta una acción o conjunto de acciones cuando se cumple una determinada condición.

Como resultado de evaluar la condición, puede ocurrir:

1. Que la condición sea VERDADERA → en cuyo caso se ejecuta la acción preestablecida
2. Que la condición sea FALSA → ante esta situación no se hace nada

Sintaxis en C:

```
if ( expresión_lógica ) {
    instrucciones;
}
```

EJERCICIOS

2.1.1. Un supermercado desea ofrecer un beneficio para atraer clientes. Decide realizar una bonificación de 15% al total de la compra si ésta es mayor o igual a $5000 y la forma de pago es en efectivo o se compran más de 10 productos (Utilice una sola expresión lógica). El ticket debe mostrar la bonificación y la forma de pago en el detalle. Las formas de pago disponibles son (c-tarjeta de crédito / d-tarjeta de débito / e-efectivo).

**Nota:*

- Prestar atención al ingreso de datos de tipo char

- Reutilizar la solución del programa correspondiente al ejercicio 1.a)

Ejemplos:

```
Cant. unidades: 7 - Precio unit.: $ 1000
Total: $ 7000.00
Forma de pago: c
Bonificación: $ 0.00
Total a pagar: $ 7000.00
¡Gracias por su compra!
```

```
Cant. unidades: 7 - Precio unit.: $ 1000
Total: $ 7000.00
Forma de pago: e
Bonificación: $ 1050.00
Total a pagar: $ 5950.00
¡Gracias por su compra!
```

2.2. Alternativa Doble

La estructura **alternativa condicional doble** permite **elegir entre dos opciones o alternativas posibles** en función del cumplimiento o no de una determinada condición que se escribe como una **expresión lógica**.

Sintaxis en C:

```
if ( expresión_lógica ){
    instrucciones_1;
}
else {
    instrucciones_2;
}
```

EJERCICIOS

2.2.1. Para otorgar un préstamo, un banco evalúa el sueldo del solicitante. Si el sueldo es mayor o igual a $100000, se autoriza un préstamo de hasta $300000. Para sueldos inferiores, sólo se autoriza un préstamo de hasta el 75% de sus ingresos. El banco desea tener un programa que permita ingresar el monto del sueldo del solicitante y muestre en pantalla una leyenda indicando el monto de préstamo autorizado.

2.2.2. Con el objeto de conseguir fondos para el viaje de egresados, los alumnos de 6to. año organizaron un baile. El precio de las entradas es de $750 con una consumición, y de $500 sin consumición. Si la venta es anticipada, se realiza un descuento del 20% al valor de la entrada. Se requiere un programa para la emisión de la entrada. La entrada debe mostrar si la venta fue anticipada ('S' / 'N'), el tipo de entrada (1-con consumición, 2-sin consumición) y el importe a pagar.
**Nota: prestar atención al ingreso de datos de tipo char.*

2.3. Alternativas Anidadas

EJERCICIOS

2.3.1. Se dispone de un termómetro para medir con exactitud la temperatura en un determinado lugar. Se desea una aplicación que permita ingresar un valor de temperatura y muestre un mensaje que indique la sensación térmica, considerando los rangos siguientes:

Rango de temperatura	Sensación térmica
[-10, 10)	Mucho frío
[10, 15)	Poco frío
[15, 25)	Temperatura normal
[25, 30)	Poco calor
[30, 45)	Mucho calor

Por ejemplo: SI (temp<10) → "Mucho frío"

**Nota:*
- *Un intervalo **cerrado** incluye los extremos. Se representa con corchetes.*
  Ejemplo: [0, 1] → todos los números mayores o iguales que 0 y menores o iguales que 1.

- *Un intervalo **abierto** no incluye los extremos. Se representa con paréntesis.*
  *Ejemplo: (-3, 3) → conjunto de números **entre** -3 y 3, sin incluir -3 y 3*

2.4. Alternativa Múltiple

En ocasiones ocurre que se necesitan más de dos alternativas.

Si bien esta situación podría resolverse anidando estructuras alternativas, esto dificulta mucho la **legibilidad** del código cuando las alternativas son numerosas. Además, presenta mayor posibilidad de cometer errores en la codificación.

La estructura **alternativa múltiple** es una toma de decisión especializada que permite **evaluar una expresión con n posibles resultados**, y en base al resultado seleccionar el siguiente bloque de instrucciones a ejecutar, de entre varios posibles.

Sintaxis en C:

```
switch ( expresión ) {
        case etiqueta1: instrucciones_1;
                        break;
        case etiqueta2: instrucciones_2;
                        break;
        default: instrucciones_n;
}
```

1. Dependiendo del valor obtenido al evaluar la expresión, se ejecutará un bloque de instrucciones u otro.
2. En las listas de valores (etiquetas) se deben escribir los valores que determinan el bloque de instrucciones a ejecutar, teniendo en cuenta que un valor sólo puede aparecer una vez en una lista de valores o etiquetas.
3. Opcionalmente, se puede escribir un bloque de *<instrucciones_n>* por defecto.

   → Este bloque de instrucciones se ejecutará en el caso de que el valor obtenido al evaluar la expresión no se encuentre en ninguna de las etiquetas especificadas de la lista de valores.

Ejemplo:

```c
#include <stdio.h>

int main() {
    short dia = 3;
    switch ( dia )
    {
    case 1:
        printf("Lunes\n");
        break;
    case 2:
        printf("Martes\n");
        break;
    case 3:
        printf("Miercoles\n");
        break;
    case 4:
        printf("Jueves\n");
        break;
    case 5:
        printf("Viernes\n");
        break;
    case 6:
        printf("Sabado\n");
        break;
    case 7:
        printf("Domingo\n");
        break;
    default:
        printf("Valor invalido\n");
    }

    return 0;
}
```

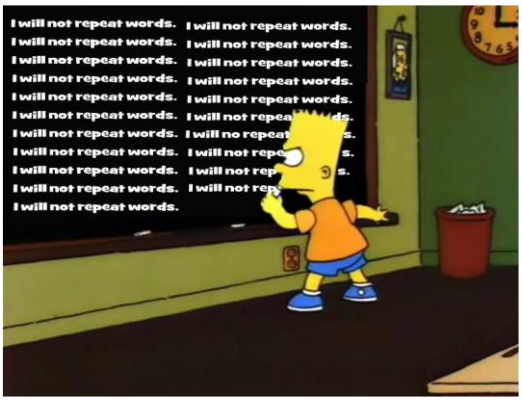
Funciona de la siguiente manera:

4. Se busca el valor de la expresión (también conocida como selector) en la lista de valores (etiquetas).
5. Si aparece, entonces se ejecuta la sentencia correspondiente.
6. Si no aparece y hay una cláusula **default** entonces se ejecuta la sentencia que le sigue.
7. Si no aparece y no hay **default** entonces no se hace nada (continua con la siguiente instrucción).

EJERCICIOS

2.4.1. Una farmacia desea emitir el ticket de compra en función de la cantidad y precio del producto vendido, y el tipo de producto (P-Perfumería, F-Farmacia y L-Limpieza). Necesita una aplicación para calcular el valor de la compra, aplicando una bonificación de acuerdo con el tipo de producto: Perfumería 5%, Farmacia 10%, Limpieza 15%. El ticket debe mostrar el importe a pagar y el porcentaje de bonificación, con la leyenda "% bonificado: XX".
**Nota: Recordar el carácter de escape.*

3. ESTRUCTURAS REPETITIVAS



3.1. Repetición Simple o Indexada (PARA/for)

En ocasiones se requiere ejecutar una secuencia de instrucciones un número determinado de veces que se conoce de antemano.

El lenguaje C posibilita automatizar tareas repetitivas de este tipo a través del comando **for**

En primer lugar, tenemos que identificar la secuencia de instrucciones que se desea repetir → identificar el **PATRÓN**.

Sintaxis en C:

```
for (inicialización; condiciónIteración; incremento) {
   secuencia de instrucciones que se desea repetir
}
```

Ejemplo: Si quisiéramos simular el desafío **No me canso de rebotar** de PilasBloques, en C podríamos escribir algo como lo siguiente:

```c
#include <stdio.h>

void rebotar30Veces();

int main(){
    printf("Tengo que hacer 30 rebotes!\n");
    rebotar30Veces();
    return 0;
}

void rebotar30Veces() {
    int i;
    for (i=0; i<30; i++)
    {
        printf("Faltan %d rebotes!\n", 30-i);
    }
    printf("Ya hice los rebotes necesarios!\n");
}
```

EJERCICIOS

Utilizando la estructura de repetición simple, diseñe en cada caso un algoritmo para:

3.1.1. Mostrar los números enteros desde 1 hasta N, sumarlos e informar la suma.

3.1.2. Calcular e informar el promedio de los números impares menores o iguales a 20.

**Nota: En este programa es conveniente declarar una constante para el valor 20. En C, una forma de declarar y definir una constante es usando la directiva #define seguido de un nombre (en el ejemplo: N) y luego de un espacio, el valor que se desea asignar a la constante (en el ejemplo: 20):*

```c
#include <stdio.h>
#define N 20
```

3.1.3. Generar y mostrar la tabla de multiplicar de un número introducido por el teclado.

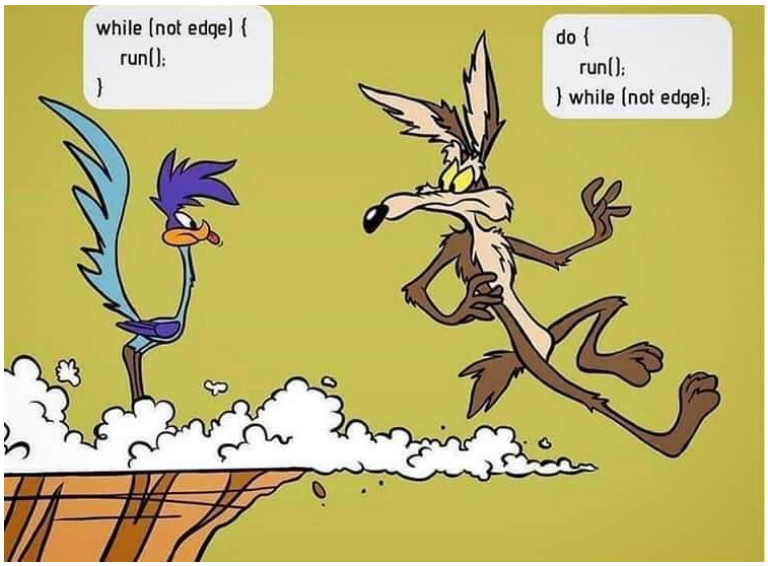
```
Introduzca un numero entero: 7

        ------------
        Tabla del  7
        ------------
        7 X 0 = 0
        7 X 1 = 7
        7 X 2 = 14
        7 X 3 = 21
        7 X 4 = 28
        7 X 5 = 35
        7 X 6 = 42
        7 X 7 = 49
        7 X 8 = 56
        7 X 9 = 63
        7 X 10 = 70
```

3.2. Repetición Condicional

Existen otras ocasiones, en que se requiere ejecutar una secuencia de instrucciones un número determinado de veces que NO se conoce de antemano.

El lenguaje C posibilita automatizar tareas repetitivas de este tipo a través de los comandos

while y **do while**


while (not edge) {
run();
}
do {
run();
} while (not edge);

WHILE

1. La condición o expresión lógica se evalúa antes de cada ejecución del bucle.

   → Si la condición es verdadera, se ejecuta el bucle, y si es falsa, el control pasa a la instrucción siguiente al bucle.

2. Si la condición se evalúa a falso cuando se ejecuta el bucle por primera vez, el cuerpo del bucle no se ejecutará nunca.

   → En este caso, se dice que se ha ejecutado **0** veces.

3. Mientras la condición sea verdadera, el bucle se ejecutará.

   → Esto significa que el bucle se ejecutará indefinidamente a menos que "algo" en el interior del bucle modifique la condición haciendo que la condición sea falsa.

→ Si la condición nunca cambia de valor, entonces el bucle no termina nunca. En este caso, se trata de un bucle infinito. Deben evitarse *estas situaciones.*

Sintaxis en C:

```
while ( condición ) {
    secuencia de instrucciones que se desea repetir
}
```

Ejemplo:

Recordemos el caso del ejercicio "Largos cambiantes" de PilasBloques en el que Yvoty necesita despertar a las luciérnagas para tener una buena iluminación, teniendo en cuenta que el camino que recorre puede ser más corto o más largo. Si lo simulamos en C podría ser algo como lo siguiente:

```c
#include <stdio.h>
#include <stdbool.h>

void despertarTodasLasLuciernagas();

int main(){
    despertarTodasLasLuciernagas();
    return 0;
}

void despertarTodasLasLuciernagas() {
    bool llegueAlfinal = false;

    while (!(llegueAlfinal == true))
    {
        printf("Continuar avanzando ...\n");
        printf("Despertar luciernaga ...\n");
        printf("Llegue al final? ");
        scanf("%d", &llegueAlfinal);
    }
}
```

**Nota: Observe que cada vez que el usuario responda a la pregunta de si llegó al final o no, debería responder ingresando el valor 1 (verdadero/true) en caso que quisiera indicar que llegó al final y caso contrario el valor 0 (falso/false).*

DO WHILE

1. La condición (expresión lógica) se evalúa al final del bucle, después de ejecutarse todas las sentencias.
2. Si la expresión lógica es **VERDADERA**, se vuelve a repetir el bucle y se ejecutan todas las sentencias.
3. Si la expresión lógica es **FALSA**, se sale del bucle y se ejecuta la siguiente instrucción luego del while.
4. Si la condición se evalúa a falso cuando se ejecuta el bucle por primera vez, el cuerpo del bucle se ejecutará al menos una vez.

Sintaxis en C:

```
do {
   secuencia de instrucciones que se desea repetir
} while ( condición );
```

Ejemplo: Ahora veamos el mismo caso del ejemplo anterior, pero aplicando el *do while*

```c
#include <stdio.h>
#include <stdbool.h>

void despertarTodasLasLuciernagas();

int main(){
    despertarTodasLasLuciernagas();
    return 0;
}

void despertarTodasLasLuciernagas() {
    bool llegueAlfinal = false;

    do
    {
        printf("Continuar avanzando ...\n");
        printf("Despertar luciernaga ...\n");
        printf("Llegue al final? ");
        scanf("%d", &llegueAlfinal);
    }while (!(llegueAlfinal == true));
}
```

Tratamiento de Cadenas de caracteres

Para trabajar con cadenas de caracteres se debe importar la librería: **#include <string.h>**

Algunas consideraciones:

→ Ingresar datos: En este caso hay que agregar un modificador al especificador %s (cadenas) para que tenga en cuenta los espacios en blanco entre palabras.

```
scanf("%[^\n]s", &nombre);
```

→ Imprimir datos: Es necesario utilizar el especificador %s

```
printf("El nombre ingresado es %s", nombre);
```

→ Declarar y asignar una variable de tipo cadena:

```
char nombre[30];    // declaración

strcpy(nombre, "Juan Perez"); // Para asignar valores se utiliza la función strcpy(variable, cadena);
```

→ Limpiar la cadena: Se copia una cadena de espacios en blanco en la cadena destino:
strcpy(nombre, " ");

→ Comparar cadenas: Se utiliza la función strcmp(cadena1, cadena2) que retorna el valor 0 en caso que las cadenas sean iguales:
int resultado = strcmp(nombre, "Juan Perez"); // → Si el valor que contiene nombre es "Juan Perez", entonces la función retorna 0, caso contrario retorna un valor distinto de 0.

Más adelante veremos este tema con más detalle.

EJERCICIOS

3.2.1. Un supermercado desea calcular el valor total de la compra a partir de los siguientes datos de los productos comprados por un cliente: cantidad, precio unitario y descripción. Se debe tener en cuenta que un cliente puede comprar varios productos distintos. El cajero ingresará -1 como valor de cantidad para indicar que terminó de ingresar los productos de un cliente. Se debe Imprimir el importe total a pagar con el formato que se muestra a continuación:

```
******************************
TOTAL A PAGAR: $ 1800.00
******************************
¡Gracias por su compra!
```

A medida que se ingresan los datos de un producto de la compra se debe mostrar el subtotal en la pantalla, de la siguiente manera:

Ejemplo 1:

```
2 X $ 18.00
Jabón en Polvo Ala $ 36.00
```

Ejemplo 2:

```
5 X $ 12.00
Fideo Pasta Tuti $ 60.00
```

Ejemplo 3:

```
3 X $ 16.00
Tomate Triturado $ 48.00
```

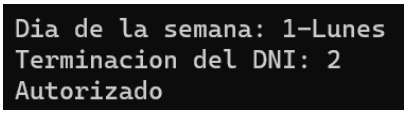
Resolver utilizando while. Realizar la verificación con los siguientes casos de prueba. Completar previamente la columna de resultado esperado:

Producto	Cantidad	Precio	Resultado esperado
Jabón Líquido Ala Matic	2	$ 860,00	$
Fideos Tallarines Canale	5	$ 120,00	$
Tomates Peritas Triturados	3	$ 150,00	$
			TOTAL A PAGAR: $

3.3. EJERCICIOS INTEGRADORES

- Atención: recordar la buena práctica de programación "Minimizar el uso de variables globales". Para aplicar esta buena práctica, utilice variables locales en todos los casos en que se trabaje con funciones.

3.3.1. Para verificar si las personas están autorizadas a circular en el marco de la pandemia, una aplicación requiere el ingreso de la terminación del DNI y el día de la semana en números (1-Lunes 2-Martes 3-Miércoles 4-Jueves 5-Viernes 6-Sábado 7-Domingo). Las terminaciones pares pueden circular los días lunes, miércoles y viernes, y las impares, los martes, jueves y sábados. El día domingo no puede circular nadie. Mostrar por pantalla los datos y una leyenda que indique "Autorizado" o "No autorizado", según corresponda, como se observa en la imagen:



Utilice una función que reciba como parámetro la terminación del DNI y devuelva verdadero si la terminación es par y falso, en caso contrario. La aplicación debe continuar funcionando hasta que se indique lo contrario.

3.3.2. La asignatura Algoritmos 1 al final del dictado cuenta con las notas del 1er. y 2do. parcial de todos sus alumnos. Se requiere determinar la categoría que le corresponde a cada uno de ellos: "Promocionó", "Regularizó", o "Libre". Se cuenta con los datos nro. de libreta universitaria y las notas el primer y segundo parcial. El parcial se aprueba con nota >= 6. Las condiciones para cada categoría son:

- Para promocionar: aprobados los 2(dos) parciales y el promedio de ambos debe ser mayor o igual a 7 (siete).
- Para regularizar: aprobados los 2(dos) parciales.
- Libre: si desaprueba al menos un parcial.

Para cada alumno, mostrar: libreta universitaria, las notas de ambos parciales y la categoría correspondiente.

**Nota: La solución debe incluir:*

- *Una función que devuelva true en el caso que el parcial esté aprobado y false en caso contrario.*
- *Una función que retorne el promedio de 2 notas que se reciben como parámetros.*

3.3.3. Modificar el programa del ejercicio anterior para que solo muestre los datos de los alumnos que promocionaron la materia. Mostrar en pantalla: libreta universitaria, nota p1, nota p2 y el valor del promedio redondeado, utilizando la función predefinida "redondeado hacia arriba".

```
double ceil(double x)
```

**Nota: La función ceil devuelve el entero más pequeño mayor o igual que el valor de x.*

3.3.4. En la Ciudad Autónoma de Buenos Aires (CABA), la tarifa base de un taxi es de $85,00 más $ 42,50 por cada kilómetro recorrido. Se desea calcular el importe recaudado por día de un taxista, para lo cual se dispone de la cantidad de kilómetros recorridos por cada pasajero. Al final de la jornada, mostrar la cantidad de pasajeros y el importe total recaudado.

**Nota: Utilizar una función que tome la distancia recorrida (en kilómetros) como parámetro y devuelva la tarifa total pagada por el pasajero.*

3.3.5. Un zoológico determina el precio de la entrada en función de la edad de los visitantes:

- Los niños menores a 6 años ingresan sin cargo.
- Los niños entre 6 y 12 años pagan 1500 pesos.
- Las personas de 65 años y más pagan 1700 pesos.
- La entrada general es de 2500 pesos.

La Administración necesita una aplicación que calcule el total a pagar por grupos de personas, en función de la cantidad de personas que conforman el grupo y la edad de cada uno de ellos. El ticket de la entrada debe mostrar la cantidad de personas y el importe total a pagar.

**Nota: Utilizar una función que reciba la edad como parámetro y devuelva el valor de la entrada a pagar.*

3.3.6. Un negocio minorista en línea ofrece envío expreso a una tarifa de $ 1250 para el primer artículo y $750 por cada artículo añadido. Necesita una aplicación que para cada pedido muestre por pantalla la cantidad de productos pedidos y el total de gastos de envío. El programa debe consultar al usuario si desea calcular los gastos de envío para otro pedido, y en caso afirmativo debe realizar el cálculo nuevamente.

**Nota: Utilizar una función que reciba como parámetro la cantidad de productos pedidos y devuelva el valor del gasto de envío.*

3.3.7. Un restaurante necesita una aplicación para generar el ticket con el importe a pagar por el consumo del cliente. Además del costo de lo consumido, se debe considerar el IVA (Impuesto al Valor Agregado) y la propina. El ticket debe contener el nombre del cliente y el importe total a pagar. El programa debe consultar al usuario si desea generar otro ticket, y en caso afirmativo volver a ejecutarse.

**Nota:*

- *Utilizar una función que reciba como parámetro el costo de la comida y devuelva el importe total a pagar, que incluya el IVA (21%) y la propina (10%).*
- *Formatear la salida para que el importe se muestre con dos decimales.*

3.3.8. Una tienda de mascotas necesita un programa para gestionar datos relacionados a los servicios proporcionados en el día y que emita información necesaria para el funcionamiento del local. En cada servicio ofrecido se registran los datos: número de atención, código de tipo de servicio (1-venta de alimentos, 2-atención a mascotas) e importe facturado.

Los dueños del local desean saber, en cada servicio prestado, si es atención a mascotas, que se muestre un mensaje que indique que recibió atención del veterinario. Ejemplo de salida en pantalla:

*** Nro de atención 12345 Recibió atención del veterinario! ***

Al finalizar la jornada, se debe realizar un cierre de caja e informar la cantidad de servicios prestados en el día, el importe total facturado y el monto promedio de facturación por servicio. El formato del informe debe ser el siguiente (Utilizar una constante para el título):

Tienda de mascotas - Servicios del día
Cantidad de servicios prestados en el día: 10
Importe total facturado: $ 23.000,00
Monto promedio de facturación: $2.300,00

**Nota: Utilizar una función con parámetros para calcular el monto promedio de facturación por servicio.*