# Práctica 4: Funciones. Paso de parámetros por valor

## Objetivos

- Aplicar el método de diseño descendente para construir programas modulares.
- Adquirir destreza en el paso de parámetros a subprogramas.
- Reforzar el concepto de ámbito de las variables al aplicarlo en el diseño de subprogramas.
- Diferenciar la implementación de funciones y procedimientos.
- Aprender a usar el paso de parámetros por valor como datos de entrada a un subprograma.
- Comprender la gestión dinámica de memoria en tiempo de ejecución a través de la pila de llamadas a funciones y de los registros de activación usando el depurador GDB.

## Conceptos Básicos

La construcción de subprogramas es la consecuencia directa de aplicar la técnica de diseño descendente de programas. En esta técnica, un problema complejo se descompone sucesivamente en problemas más sencillos cuya combinación resuelve el problema original. De esta forma, cada subproblema es convertible en un subprograma de forma directa y sencilla.

A los subprogramas que devuelven un valor se les denomina **FUNCIONES.** Si el subprograma no devuelve ningún valor, se le denomina **PROCEDIMIENTO.**

El nombre del subprograma es utilizado en tres sitios diferentes de un programa. En primer lugar, aparece en el **prototipo del subprograma**. Éste indica al programa que existe un subprograma con un nombre determinado, que devuelve un cierto tipo de dato y cuántos parámetros tiene (número y tipo). En segundo lugar, aparece dentro de la función main() (o dentro de otros subprogramas), son las **llamadas al subprograma**. En tercer lugar, en la parte donde se define el código del subprograma, la **definición del subprograma**.

**Prototipo de un subprograma**.

El prototipo de un subprograma tiene en C++ la siguiente sintaxis:

```
<tipo_devuelve> nombre_subprograma (<tipo_param1>, <tipo_param2>, . . .);
```

Algunos ejemplos de prototipos:

Una función que devuelve el máximo de dos números que se le pasan como datos de entrada.

```
float maximo (float, float);
```

Una función que muestra un menú y devuelve la opción elegida por el usuario.

```
unsigned short menu ();
```

Una función que calcula la edad de una persona conocido el año de nacimiento.

```
unsigned short edad (unsigned short);
```

**Codificación de un subprograma. Sintaxis.**

```
<tipo_devuelve> nombre_subprograma (<tipo_param1> nombre_param1, . . .)
{
    // DECLARACION DE VARIABLES LOCALES
    . . .

    // ZONA DE CODIGO
    . . .

    return <valor_retorno>  // PROCEDIMIENTO: no es obligatorio
}
```

**Cuidado:** Si no colocamos nada en `tipo_devuelve`, la función por defecto devuelve un entero. Si queremos indicar que no devuelve nada (un procedimiento), el tipo devuelto es `void`.

**Ejemplos:**

**Función que calcula el máximo de dos números:**

```
float maximo (float n1, float n2)
{
  float aux;


  if (n1 > n2)
     aux = n1;
  else
     aux = n2;

  return aux;
}
```

**Función que calcula la edad conocido el año de nacimiento:**

```
unsigned short edad (unsigned short anyo)
{
  unsigned short res;


  res = 2021 - anyo;

  return res;
}
```

**Procedimiento que imprime la secuencia de números menores que el número que se pasa como parámetro:**

```
void sec_numeros (unsigned int num)
{
  unsigned int i;

  for (i = 1; i < num; i++)
    cout << i << endl;
}
```

**Procedimiento que imprime la tabla de multiplicar del número que se pasa como parámetro:**

```
void TablaMultiplicar (unsigned short n)
{
  unsigned short i;

  for (i = 1; i <= 10; i++)
     cout << i << "*" << n << "=" << n * i << "\n";
}
```

Las funciones pueden formar parte de una expresión, es decir, se puede usar como un operando en la expresión.

Los procedimientos son sentencias, de forma que al no devolver nada no se puede: ni usar como parte derecha de una asignación, ni aplicar el operador de salida (<<), ni como operando en una expresión.

**Llamada a un subprograma**.

Un subprograma puede ser llamado desde cualquier punto del programa. Por ejemplo: en la línea 08 se hace una llamada la función `maximo` pasándole dos parámetros a y b.

```
01 int main ()
02 {
03     float a, b, result;
04
05     cout << "Introduce dos valores:\n";
06     cin >> a >> b;
07
08     result = maximo(a, b);
09
10     cout << result;
11
12     return 0;
13 }
```

**Parámetros formales y reales**.

Por parámetros formales entendemos los parámetros que se utilizan en la codificación del subprograma. Por ejemplo, en la función `maximo` los parámetros n1 y n2 reciben el nombre de **parámetros formales**. En cambio, cuando la función es llamada en la línea 08, se les da un valor concreto y son denominados **parámetros reales**.

El identificador de los parámetros formales y reales puede coincidir o no, eso no implica que sean la misma variable. Recuerda que una variable en programación tiene más atributos aparte de su identificador: el valor, el tipo de dato, el ámbito y el tiempo de vida.

**Ámbito de las variables**.

Tanto las variables declaradas dentro del subprograma como los parámetros formales tienen un **ámbito local** al subprograma (el trozo de código donde puede ser usado es el cuerpo del subprograma). Ningún otro subprograma puede acceder a las variables locales de un subprograma. A este mecanismo se le llama **encapsulación** de los datos. Cualquier intento de acceder a variables declaradas en otros subprogramas generará un error al compilar.

Una variable global tiene validez en todo el programa y puede ser usada en todos los subprogramas, salvo que exista una variable local que sea idéntica en nombre y tipo. En este caso, la variable local tiene precedencia sobre (**oculta a**) la variable global en el subprograma.

**Estructura de un programa. Ejemplo**:

```
/* ZONA DE LOS INCLUDES */
#include <iostream>
. . .

using namespace std;


/* ZONA DE DECLARACIÓN DE COSTANTES */
const float PI = 3.1415;


/* ZONA DE DECLARACION DE VARIABLES GLOBALES */
float x;


/* ZONA DE DECLARACION DE LOS PROTOTIPOS DE LAS FUNCIONES */
int sumar (int, int);


/*CODIFICACIÓN DE LA FUNCIÓN PRINCIPAL */
int main ()
{
  . . .
}

/* CODIFICACIÓN DEL RESTO DE FUNCIONES */
int sumar (int a, int b)
{
  . . .
}
```

**Paso de parámetros por valor.**

Cuando en la llamada a un subprograma pasamos un **parámetro por valor**, se hace **una copia del valor que toma el parámetro real al parámetro formal** del subprograma. Se reserva memoria para el parámetro formal y se guarda una copia del valor que tiene el parámetro real. Una vez acabado el subprograma se libera la memoria. Cualquier cambio en el valor del parámetro formal **no altera** el valor del parámetro real (son variables distintas).

**Recuerda**:
Las funciones no leen datos de entrada desde teclado ni muestran resultados por pantalla. Las funciones tienen parámetros (de entrada y/o salida) y devuelven resultados. No tiene sentido usar dentro de las funciones los operadores de entrada y de salida de flujo: >> y <<.

**Excepción**:
La tarea a realizar por la propia función es la lectura de datos desde teclado o la impresión de resultados por pantalla.

## BLOQUE DE EJERCICIOS

**Ejercicio 1 (ejercicio1.cpp):** Analiza el siguiente programa y modifícalo hasta que funcione correctamente.

```
#include <iostream>

using namespace std;

numeroImpar (                    ); // línea a modificar

int main ()
{
    int valor_leido;
    bool res;

    cout << "Introduce un valor: ";
    cin >> valor_leido;

    numeroImpar (              ); // línea a modificar

    if (res == true)
       cout << valor_leido << " es impar" << endl;
    else
       cout << valor_leido << " es par" << endl;

    return 0;
}
```

```
// Esta función determina si el número que se le pasa como parámetro es impar
bool numeroImpar (                        )  // línea a modificar
{
    bool res = false;

    if (valor % 2 != 0)
       res = true;

    return res;
}
```

**Ejercicio 2 (Depurando un programa, ejercicio2.cpp)**. Abre el fichero ejercicio2.cpp. Compílalo y ejecútalo.

**Parte 2a.**

Comprueba si el resultado es correcto o no para los siguientes valores:

| m | n | Valor real | Valor calculado | ¿Correcto? |
|---|---|---|---|---|
| 6 | 2 | | | |
| 10 | 2 | | | |
| 15 | 2 | | | |

https://www.omnicalculator.com/math/binomial-coefficient

https://www.calculatorsoup.com/calculators/discretemathematics/factorials.php

**Parte 2b.**

Veamos qué ocurre paso a paso. Establece puntos de ruptura en las líneas 45, 49, 70, 90 y 94. Para conocer el contenido de la pila de llamadas a funciones (`call stack`) en esas líneas usa los siguientes comandos del depurador en el cuadro de texto: `Send command to GDB` . Avanza paso a paso para aplicar los comandos del depurador cuando desees, `Next line`.

| | |
|---|---|
| `info locals` | muestra el valor que almacenan las variables locales del registro de activación actual (él que se encuentra en la cima). https://visualgdb.com/gdbreference/commands/info_locals |
| `info args` | muestra el valor que almacenan los parámetros formales del registro de activación actual. https://visualgdb.com/gdbreference/commands/info_args |
| `frame` | muestra información sobre el registro de activación actual. Presta especial atención a la información que se imprime: **frame-function-name, arg-begin, arg-name-end, arg-value** https://visualgdb.com/gdbreference/commands/frame |
| `print var_name` | muestra el valor de una variable del registro de activación actual. https://visualgdb.com/gdbreference/commands/print |

Para `m = 6` y `n = 2`:

i) Dibuja la cadena de la jerarquía de llamadas a funciones (ver diapositivas 39-41).

ii) Indica el contenido de la pila con los registros de activación de las llamadas a funciones (valor de los parámetros formales, valor de las variables locales y valor de retorno) en las líneas 45, 49, 70, 90 y 94, cada vez que el depurador las visite (ver diapositivas 39-47).

Rellena la siguiente tabla para los siete instantes de tiempo.

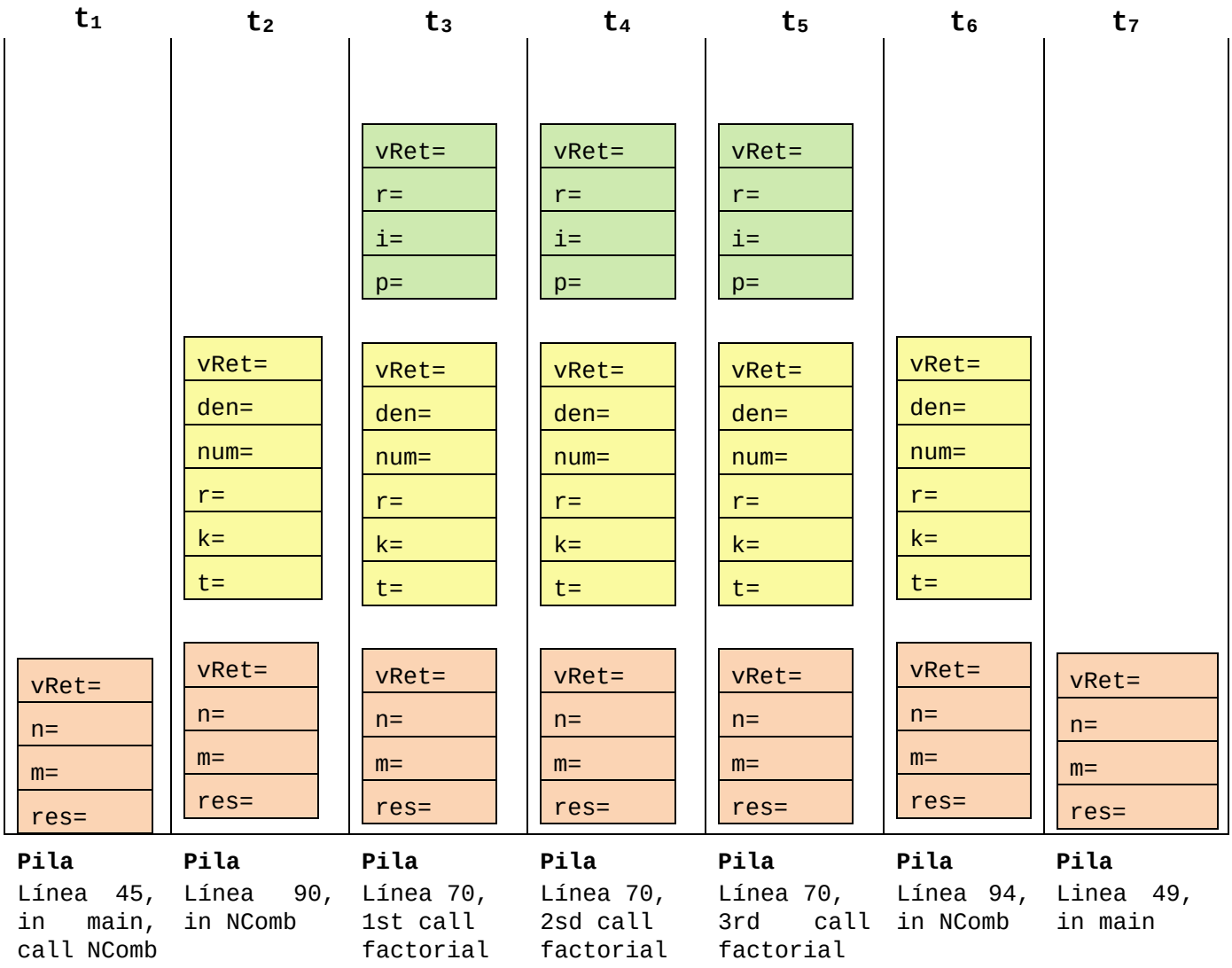


**Tabla 2b. Caso m = 6, n = 2.**

Para cada instante de tiempo indica la cantidad de variables declaradas diferentes que hay almacenadas en la pila. Justifica la respuesta. Calcula el tamaño en bytes de los registros de activación de cada una de las funciones que se están ejecutando en ese instante, incluye también el valor de retorno. Usa la función `sizeof (unsigned int)` para obtener el número de bytes que ocupa este tipo de dato en tu ordenador.

| Instante | $t_1$ | $t_2$ | $t_3$ | $t_4$ | $t_5$ | $t_6$ | $t_7$ |
|---|---|---|---|---|---|---|---|
| #vars distintas | | | | | | | |
| #bytes Factorial | | | | | | | |
| #bytes NCombinatorio | | | | | | | |
| #bytes main | | | | | | | |
| #bytes total pila | | | | | | | |

Para `m = 15` y `n = 2`:

i) Dibuja la cadena de la jerarquía de llamadas a funciones.

ii) Construye un tabla similar a la del apartado anterior para este caso Tabla 2b. Caso m = 15, n = 2.

**Parte 2c.**

**¿Cuál es el error que se ha cometido en la implementación?** Corrígelo y comprueba que ahora la información que almacena la pila para m = 15 y n= 2 es correcta.

*Ayuda***:** Haz un pequeño programa para visualizar la cantidad de bytes que utilizan los siguientes tipos de datos y el valor máximo que pueden almacenar. Y sabrás cómo solucionarlo. Incluye <climits>**.**

```
cout << sizeof(unsigned int) << endl;
cout << sizeof(long long) << endl;
cout << UINT_MAX << endl;
cout << LLONG_MAX << endl;
```

**Ejercicio 3 (ejercicio3.cpp)**. Escribe una función, **Comparar**, que devuelva el mayor de dos números reales que se le pasan como parámetros de entrada. Escribe un programa en C++ donde se pida al usuario cuatro números reales y muestre por pantalla el mayor de los cuatro. *Ayuda:* Utiliza varias veces la función anterior comparando números de dos en dos.

**Ejercicio 4 (ejercicio4.cpp).** Escribe una función, **InvertirNumDigitos**, que reciba un valor entero positivo como parámetro de entrada y devuelva un número con sus dígitos invertidos. Implementa una función, **NumCapicua**, que, usando la función anterior, determine si un número entero positivo que se le pasa como parámetro de entrada es o no capicúa.

Realiza un programa en C++ que lea un número entero positivo desde el teclado y muestre por pantalla el número con las cifras al revés. En caso de que el número sea capicúa se debe imprimir el correspondiente mensaje. **El proceso se puede repetir mientras el usuario lo desee**.

Un ejemplo de ejecución es:

```
Introduce un número: 143
El número 143 con los dígitos invertidos es 341.

Desea continuar (S/N)? S

Introduce un número: 515
El número 515 con los dígitos invertidos es 515.
El número es capicúa. ¡Suerte!.

Desea continuar (S/N)? N
Bye-Bye.
```

**Ejercicio 5 (ejercicio5.cpp).** Realiza un programa en C++ que genere y muestre por pantalla una secuencia de números aleatorios en un intervalo. Implementa la función y el procedimiento siguiente:

- La función **NumAleatorio** tiene dos parámetros de entrada (los límites del intervalo) y devuelve un número aleatorio en el intervalo (ver Cuestión 4 del documento de prerrequisitos).

- El procedimiento **SecuenciaAleatoria** tiene tres parámetros de entrada: la cantidad de números a generar y los límites del intervalo. En este procedimiento se inicializa la semilla de números aleatorios, se llama a la función anterior tantas veces como sea necesario y se imprimen cada uno de los números generados.

La función principal lee desde teclado la cantidad de números de la secuencia y los límites del intervalo y llama al procedimiento **SecuenciaAleatoria** para mostrar la secuencia de números.

**Se debe garantizar** que la cantidad de números de la secuencia es mayor que cero y que el límite superior del intervalo es mayor que el límite inferior. De lo contrario, se pedirá al usuario de nuevo estos valores hasta que sean correctos.

**IMPORTANTE:**

- Todos los programas deben seguir la Guía de Estilo e incluir los comandos de Doxygen para generar la documentación.
- Subir al Aula Virtual los archivos .cpp de uno en uno (enunciado y adicionales). Sólo los sube un miembro de la pareja.
- Fecha de entrega: durante la sesión de vuestro grupo de lab.