Temario:

- Recursividad

Aclaración: en los ejercicios donde se pide escribir funciones, agregar los parámetros (con o sin valores por defecto) necesarios.

Ejercicios:

Ejercicio 1.

Dado el siguiente programa, analizarlo (sin ejecutarlo en la máquina) e indicar qué calcula. ¿Cuál es el caso base de este algoritmo? ¿Y el caso recursivo?

```
#include <iostream>
using namespace std;

int funcion(int m, int n) {
   if (n == 0) {
       return 0;
   }
   return m + funcion(m, n-1);
}
int main() {
    cout << funcion(20, 3);
    return 0;
}
```

Ejercicio 2.

Graficar la pila de llamadas (colocando parámetros y valores de retorno):

```
int sumatoria(int num) {
    if (num == 0)
        return 0;
    return sumatoria(num-1) + num;
}
```

```
int main() {
    cout << sumatoria(5);
    return 0;
}
```

Ejercicio 3.

Escribir una función recursiva que, dado un número entero, lo imprima en orden inverso.

Ejemplo: si el número es 5182, la función debe imprimir 2815.

Ejercicio 4.

Escribir una función recursiva que, dado un número entero, retorne la cantidad de dígitos que tiene.

Ejemplo: si el número es 650, la función debe retornar 3.

Ejercicio 5.

Escribir una función recursiva que calcule la suma de los cuadrados de los N primeros números positivos.

Ejemplo: si N=4 la suma es $1^2 + 2^2 + 3^2 + 4^2 = 30$.

$$suma(N) = \begin{cases} 1 & \text{si } N = 1 \\ N^2 + suma(N-1) & \text{en caso contrario} \end{cases}$$

Ejercicio 6.

Escribir una función recursiva que calcule el producto de dos números naturales, usando sumas.

Ejemplo: 2*3 se puede calcular como 2+2+2.

Ejercicio 7.

Escribir una función recursiva para calcular la potencia de un número. La función recibirá como parámetros al número y un exponente (ambos enteros positivos), y retornará la potencia:

```
int potencia(int numero, int exponente)
```

Nota 1: Cualquier número elevado a la 0 da como resultado 1. El 0 elevado a cualquier número positivo dará siempre como resultado 0.
Nota 2: Puede definirse que $X^n = X * X^{n-1}$

Ejercicio 8.

Escribir una función recursiva que reciba un string y lo retorne con sus caracteres invertidos.

Nota: Podría ser útil la función *substr* que obtiene un substring a partir de un string.

Ejercicio 9.

Escribir una función recursiva que reciba una palabra y retorne true si es palíndromo, false si no lo es (diferenciando mayúsculas y minúsculas).

Ejemplo: si la palabra es "radar" retorna true.
Nota 1: Un palíndromo es una palabra que se lee igual en un sentido que en otro.
Nota 2: Podría ser útil la función *substr* que obtiene un substring a partir de un string.

Ejercicio 10.

Indicar qué imprime el siguiente programa (sin ejecutarlo en la máquina). ¿Cuál sería un nombre adecuado para la función llamada "f"?:

```
int min(int a, int b) {
   if (a < b)
       return a;
   else
       return b;
}

int f(int arreglo[], int dl) {
```

```
   if (dl == 1)
       return arreglo[0];
   else
       return min(arreglo[dl-1], f(arreglo, dl-1));
}

int main() {
   int a[] = { 6,2,3,7,9,4 };
   int dl = 6;
   cout << f(a, dl);
}
```

Ejercicio 11.

Escribir una función recursiva que reciba un arreglo de int y su dimensión lógica, y retorne el primer número impar encontrado en el arreglo. En caso de no haber números impares, debe retornar 0.

Ejemplo: Dado el arreglo [2, 0, 8, 7, 4, 3] retornará 3.

Ejercicio 12.

Escribir una función recursiva que reciba un arreglo de int y su dimensión lógica, y retorne la suma de los elementos pares almacenados en el arreglo. En caso de no haber números pares, debe retornar 0.

Ejemplo: Dado el arreglo [4, 5, 10, 17, 2, 3] retornará 16.

Ejercicio 13.

a) Suponer la existencia de una lista de números enteros positivos cuyos nodos se definen como:

```
struct Nodo {
    int dato;
    Nodo* siguiente;
};
```

Analizar la siguiente función (sin ejecutarla en la máquina) e indicar qué retorna si la lista pasada por parámetro contiene los siguientes elementos: 6 - 2 - 8 - 1.

```
int funcion(Nodo* inicio, int m = -1) {
   if (inicio == nullptr)
```

```
        return m;
    if (inicio->dato > m)
        m = inicio->dato;
    return funcion(inicio->siguiente, m);
}
```

b) ¿Qué debería cambiarse en la función anterior si se intentara obtener el resultado opuesto? En el ejemplo de la lista con los números 6 - 2 - 8 - 1 se debería obtener el número 1. Se asume que la lista no contiene valores mayores que 99.

Ejercicio 14.

Analizar el programa siguiente (sin ejecutarlo en la máquina) e indicar cómo queda la lista luego de la ejecución.

```
struct Nodo {
   int dato;
   Nodo* siguiente;
};

Nodo* e(Nodo* inicio, int k) {
   if (k < 1)
      return inicio;
   if (inicio == nullptr)
      return nullptr;
   if (k == 1) {
       Nodo *aux = inicio->siguiente;
       delete(inicio);
       return aux;
   }
   else {
       inicio->siguiente = e(inicio->siguiente, k-1);
       return inicio;
   }
}

Nodo* a(Nodo* inicio, int d) {
   if (inicio == nullptr) {
       Nodo* nuevo = new Nodo;
```

```
        nuevo->dato = d;
        nuevo->siguiente = nullptr;
        return nuevo;
    }
    else {
        inicio->siguiente = a(inicio->siguiente, d);
    }
    return inicio;
}

int main() {
    Nodo* inicio=nullptr;
    inicio = a(inicio, 9);
    inicio = a(inicio, 6);
    inicio = a(inicio, 4);
    inicio = a(inicio, 7);
    inicio = e(inicio, 2);
}
```

Ejercicio 15.

Escribir una función recursiva que, dada una lista de enteros, retorne la suma de todos sus elementos. En caso de que la lista esté vacía, deberá retornar 0.

Ejemplo: si la lista contiene 5 - 2 - 1 - 9 retornará 17.

Ejercicio 16.

Escribir una función recursiva que, dadas dos listas enlazadas simples, retorne true si ambas listas tienen la misma longitud (cantidad de nodos), false en caso contrario. Optimizar la función para que, en caso de que una lista sea más corta que la otra (por ejemplo, la lista A tiene 10000 nodos y la lista B tiene 5 nodos), no se continúe contabilizando la cantidad de nodos de la lista más larga.