UNIVERSIDAD AMERICANA
*Facultad de Ingeniería y Arquitectura*

---


# *Libre*

---

*Documentación: Módulos y arreglos*

---

**Estudiantes:**

- Arelys Brigitte Obando Castillo
- Kimberly Maria Zapata Espinoza

**Docente:**

- Silvia Gigdalia Ticay López

*Managua, 2024*

**Ejercicios de evaluación S9: Módulos y arreglos**

**Ejercicio 1.** Programa que sea capaz de almacenar los datos de 10 personas: nombre, dirección,
teléfono, edad (usando structs). Deberá ir pidiendo los datos uno por uno, hasta que el usuario lo decida. Entonces deberá aparecer un menú que permita:

• Mostrar la lista de todos los nombres.
• Mostrar las personas de una cierta edad.
• Mostrar las personas que coincidan con un nombre. Sea el que el usuario indique.
• Salir del programa.

- Cada opción del menú representa una función o procedimiento que se invocará desde la clase principal.
- Utilizar funciones o procedimientos con parámetros.

```
static void Main(string[] args)
{
    //lista para almacenar objetos de tipo Estudiante
    List<Estudiante> Estudiantes = new List<Estudiante>();
    int opcion;

    // Ciclo que se repite hasta que el usuario elige salir
    do
    {
        //menú principal
        Console.WriteLine("\nGestión de Estudiantes");
        Console.WriteLine("1. Añadir estudiante");
        Console.WriteLine("2. Mostrar todos los nombres");
        Console.WriteLine("3. Mostrar por edad");
        Console.WriteLine("4. Mostrar por nombre");
        Console.WriteLine("5. Salir");
        Console.Write("Selecciona una opción: ");
        opcion = int.Parse(Console.ReadLine());
```

1. Creamos una lista que almacena los objetos de tipo Estudiante (List<Estudiante> Estudiantes) y una variable para almacenar la opción seleccionada por el usuario en el menú.
2. El programa utiliza un ciclo do-while para mostrar un menú de opciones hasta que el usuario decida salir.

```
    switch (opcion)
    {
        case 1:
            AgregarEstudiante(Estudiantes);
            break;
        case 2:
            MostrarNombres(Estudiantes);
            break;
        case 3:
            MostrarPorEdad(Estudiantes);
            break;
        case 4:
            MostrarPorNombre(Estudiantes);
            break;
        case 5:
            Console.WriteLine("\nSaliendo del programa.");
            break;
        default:
            Console.WriteLine("\nOpción inválida. Intente de nuevo.");
            break;
    }
    Console.ReadKey();
    // Limpia la consola para mostrar el menú nuevamente
    Console.Clear();
while (opcion != 5);
```

3. Dentro del ciclo, se presenta un menú al usuario y se utiliza un switch para manejar la opción seleccionada. El programa se cerrará en el caso de que la opción seleccionada sea 5. En el caso de ingresar un número fuera de los casos establecidos se imprimirá en consola “Opción Inválida. Intente de nuevo”. Además, luego de realizar cada función se limpiará la consola.

**Funciones y la estructura de la clase FuncionesYmetodos**
**Struct**

```
public class Estudiante
{
    //Propiedades para almacenar
    public string Nombre;
    public string Direccion;
    public int Telefono;
    public int Edad;

    //Constructor
    1 referencia
    public Estudiante(string nombre, string direccion, int telefono, int edad)
    {
        Nombre = nombre;
        Direccion = direccion;
        Telefono = telefono;
        Edad = edad;
    }
}
```

4. La clase Estudiante representa la información de un estudiante. Donde las propiedades almacenan los datos ingresados por el usuario y el constructor los parámetros.

**Métodos**
**AgregarEstudiante(List<Estudiante> Estudiantes) para el Case 1 del Switch**

```
public static void AgregarEstudiante(List<Estudiante> Estudiantes)
{
    // Verifica si ya se alcanzó el límite de 10 estudiantes
    if (Estudiantes.Count >= 10)
    {
        Console.WriteLine("\nSe ha alcanzado el límite de registros.");
        return;
    }

    // Solicita los datos del nuevo estudiante
    Console.WriteLine("\nRegistrar Estudiantes");
    Console.Write("Nombre: ");
    string nombre = Console.ReadLine();
    Console.Write("Dirección: ");
    string direccion = Console.ReadLine();
    Console.Write("Teléfono: ");
    int telefono = int.Parse(Console.ReadLine());
    Console.Write("Edad: ");
    int edad = int.Parse(Console.ReadLine());

    // Crea un nuevo objeto Estudiante con los datos ingresados
    Estudiante nuevaPersona = new Estudiante(nombre, direccion, telefono, edad);
    // Agrega el nuevo estudiante a la lista
    Estudiantes.Add(nuevaPersona);
    Console.WriteLine("\nEstudiante añadido exitosamente.");
}
```

5. Este método permite agregar un nuevo estudiante a la lista de estudiantes. Primero verifica si ya se ha alcanzado el límite de 10 estudiantes. Si es así, muestra un mensaje y termina la ejecución. Sino, solicita al usuario que ingrese los datos del nuevo estudiante: nombre, dirección, teléfono y edad y Crea una nueva instancia de Estudiante con los datos ingresados y la agrega a la lista.
   **Parámetros**
   **List<Estudiante> Estudiantes: Lista donde se almacenan los estudiantes.**

**MostrarNombres(List<Estudiante> Estudiantes) para el Case 2 del Switch**

```
public static void MostrarNombres(List<Estudiante> Estudiantes)
{
    Console.WriteLine("\nLista de Estudiantes:");
    // Recorre la lista de estudiantes e imprime cada nombre
    foreach (var Estudiante in Estudiantes)
    {
        Console.WriteLine(Estudiante.Nombre);
    }
}
```

6. Este método imprime todos los nombres de los estudiantes almacenados en la lista. Recorre la lista de estudiantes e imprime el nombre de cada uno.
   **Parámetros**
   **List<Estudiante> Estudiantes: Lista de estudiantes de la que se mostrarán los nombres.**

**MostrarPorEdad(List<Estudiante> Estudiantes) para el Case 3 del Switch**

```
// Método para mostrar estudiantes filtrados por edad
1 referencia
public static void MostrarPorEdad(List<Estudiante> Estudiantes)
{
    Console.Write("\nIngrese la edad que desea buscar: ");
    int edadBuscada = int.Parse(Console.ReadLine());
    Console.WriteLine("Personas de {0} años:", edadBuscada);

    // Recorre la lista de estudiantes y muestra los que coinciden con la edad buscada
    foreach (var Estudiante in Estudiantes)
    {
        if (Estudiante.Edad == edadBuscada)
        {
            Console.WriteLine(Estudiante.Nombre);
        }
    }
}
```

7. Este método permite filtrar y mostrar estudiantes según una edad específica ingresada por el usuario. Luego solicita al usuario que ingrese una edad y recorre la lista de estudiantes y muestra los nombres de aquellos cuya edad coincida con la ingresada.
   **Parámetros**
   **List<Estudiante> Estudiantes: Lista de estudiantes a filtrar.**

**MostrarPorNombre(List<Estudiante> Estudiantes) para el Case 4 del Switch**

```
public static void MostrarPorNombre(List<Estudiante> Estudiantes)
{
    Console.Write("\nIngrese el nombre que desea buscar: ");
    string nombreBuscado = Console.ReadLine();
    Console.WriteLine("\nPersonas con el nombre {0}:", nombreBuscado);

    // Recorre la lista de estudiantes y muestra los que coinciden con el nombre buscado
    foreach (var Estudiante in Estudiantes)
    {
        // Comparación de nombres ignorando mayúsculas y minúsculas
        if (Estudiante.Nombre.Equals(nombreBuscado, StringComparison.OrdinalIgnoreCase))
        {
            //Imprimir
            Console.WriteLine("{0}", Estudiante.Nombre);
            Console.WriteLine("{0}", Estudiante.Direccion);
            Console.WriteLine("{0}", Estudiante.Telefono);
            Console.WriteLine("{0}\n", Estudiante.Edad);
        }
    }
}
```

8. Este método permite buscar y mostrar los datos de los estudiantes según un nombre específico ingresado por el usuario. Primero solicita al usuario que ingrese un nombre y luego recorre la lista de estudiantes y muestra los detalles (nombre, dirección, teléfono y edad) de aquellos que coincidan con el nombre ingresado, ignorando mayúsculas y minúsculas.
   **Parámetros**
   **List<Estudiante> Estudiantes: Lista de estudiantes a buscar.**

**Ejercicio 4.** Crea un programa que use un arreglo estático para almacenar números y una función que calcule el factorial de cada número, el cual es enviado a un segundo arreglo. Muestra los resultados, es decir ambos arreglos
- El número es leído en la función principal Main y es enviado como parámetro a la función que calcula el factorial. Recuerda que el factorial no se calcula para números negativos. Por lo tanto, al arreglo original sólo debes guardar los números positivos o cero.
- El programa se repetirá mientras el usuario lo desee.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Numerics; //Para usar el BigInteger
using System.Text;
using System.Threading.Tasks;

namespace Ejercicio4_evaluacion
{
    2 references
    internal class CalcularFactorial
    {
        // Función para calcular el factorial de un número
        1 reference
        public static BigInteger Factorial(int numero)
        {
            // Si el número es 0 o 1, el factorial siempre es 1
            if (numero == 0 || numero == 1)
            {
                return 1;
            }

            // Inicializamos la variable factorial con 1 para comenzar el cálculo
            BigInteger factorial = 1;

            // For que multiplica factorial por cada número desde 2 hasta el número dado
            for (int i = 2; i <= numero; i++)
            {
                factorial *= i;  // Multiplica el valor actual de factorial por i
            }

            // Devuelve el resultado final, que es el factorial del número
            return factorial;
        }
```

1. Se crea una clase externa al main, llamada **CalcularFactorial** que va a contener los métodos para calcular el factorial y mostrar los resultados.
2. El programa inicia con una condición if, se verifica si el número ingresado es igual a cero o a 1. Si es así, se devuelve 1 porque el factorial de 0 y 1 siempre es 1.
3. Se crea una variable llamada factorial, esta de tipo BigInteger, ya que si se dejaba como int, el programa no imprimía el factorial de algunos números que son muy grandes, ya que el formato de int no lo permite. A esta variable se le asigna el valor de 1 y es la que se usará para calcular el factorial.
4. Luego se establece un bucle for que comienza en 2 y continúa hasta el número ingresado (número). Este bucle se usa para multiplicar todos los números desde 2 hasta el número ingresado.
5. En cada interacción del bucle, se multiplica la variable factorial por el valor actual de i, acumulando el producto de todos los números para calcular el factorial.

6. Y luego devuelve el resultado, luego que el bucle termina se devuelve el valor de factorial, del número ingresado.

```
        // Función para mostrar los números y sus factoriales
        1 reference
        public static void MostrarResultados(int[] numeros, BigInteger[] factoriales, int cantidad)
        {
            Console.WriteLine("Número\tFactorial");
            for (int i = 0; i < cantidad; i++)
            {
                Console.WriteLine($"{numeros[i]}\t{factoriales[i]}");
            }
        }

    }
}
```

7. Se crea el procedimiento MostrarResultados, que recibe dos arreglos (números y factoriales) y un entero (cantidad).
8. Se imprime el encabezado "Número\tFactorial", y luego, en un bucle, se muestran cada número junto a su factorial usando Console.WriteLine.
9. El procedimiento termina después de imprimir todos los números y sus factoriales.

**En la clase principal:**

```
Console.WriteLine("PROGRAMA QUE CALCULA EL FACTORIAL DE NÚMEROS POSITIVOS");
const int tamaño = 100;  // Definimos el tamaño del arreglo
int[] numeros = new int[tamaño];  // Arreglo para almacenar los números positivos o cero
BigInteger[] factoriales = new BigInteger[tamaño];  // Ponemos el tipo a BigInteger para que pueda imprimir resultados grandes
int indice = 0;  // Índice para controlar la posición en el arreglo
string continuar;
```

1. Se imprime en la consola el mensaje "PROGRAMA QUE CALCULA EL FACTORIAL DE NÚMEROS POSITIVOS", informando al usuario sobre la funcionalidad del programa.
2. Se establece una constante **tamaño** con un valor de 100, que representa el número máximo de elementos que se pueden almacenar en los arreglos.
3. Se inicializan dos arreglos: **numeros** para almacenar los números ingresados (tipo int) y **factoriales** para almacenar los resultados de los factoriales (tipo BigInteger). Además, se declara una variable **índice** para rastrear la posición actual en el arreglo y una variable **continuar** para gestionar el flujo del programa.

```
do
{
    // Leer número ingresado por el usuario
    Console.Write("Ingrese un número positivo o cero: ");
    int numero = int.Parse(Console.ReadLine());

    if (numero >= 0 && indice < tamaño)
    {
        // Agregar el número al arreglo si es válido y hay espacio
        numeros[indice] = numero;
        factoriales[indice] = CalcularFactorial.Factorial(numero);  // Calcular el factorial y almacenarlo
        indice++;  // Incrementar el índice
    }
    else if (numero < 0)
    {
        Console.WriteLine("Solo se permiten números positivos o cero.");
    }
    else
    {
        Console.WriteLine("El arreglo está lleno. No puede ingresar más números.");
        break;
    }

    // Preguntar si el usuario quiere continuar
    Console.Write("¿Desea ingresar otro número? (s/n): ");
    continuar = Console.ReadLine().ToLower();
} while (continuar == "s" && indice < tamaño);
```

4. Se utiliza un bucle **do...while** para solicitar al usuario que ingrese un número positivo o cero, que se almacena en la variable **numero**.
5. Se verifica si el número es positivo y si hay espacio en el arreglo. Si es válido, se agrega al arreglo **numeros** y se calcula su factorial, que se almacena en el arreglo **factoriales**. Se incrementa el índice para el próximo número. Si el número es negativo, se informa al usuario que solo se permiten números positivos. Si el arreglo está lleno, se notifica al usuario y se sale del bucle.
6. Después de procesar el número, se pregunta al usuario si quiere ingresar otro número. Si el usuario responde con "s" (sí) y hay espacio en el arreglo, el bucle se repite; de lo contrario, se finaliza, igualmente se coloca la función **ToLower** por si el usuario ingresa la letra en minuscula o mayuscula no afecte el programa.

```
// Mostrar los resultados
CalcularFactorial.MostrarResultados(numeros, factoriales, indice);
Console.ReadKey();
```

7. Se llama a la función **MostrarResultados** de la clase **CalcularFactorial**, pasando los arreglos numeros, factoriales y el índice indice como argumentos para imprimir los números ingresados y sus respectivos factoriales.

**Ejercicio 5.** Escribe el programa que tenga una función que reciba un arreglo de enteros y lo invierta (el primer elemento se convierte en el último, el segundo en el penúltimo, etc.). Muestra el arreglo original y el invertido.

- No puedes utilizar métodos ya definidos en el lenguaje.
- Implementa una función que determine en el arreglo invertido, cuántos valores impares existen y los imprima.

```
static void Main(string[] args)
{
    //Variable para almacenar la cantidad de elementos en el arreglo
    int n;
    Console.Write("Ingrese la cantidad de elementos en el arreglo: ");
    n = int.Parse(Console.ReadLine());

    // Crea un arreglo de enteros con la longitud especificada por el usuario
    int[] numeros = new int[n];
```

1. En el punto de partida del programa declaramos las variables **int n** que almacena la cantidad de elementos que el usuario desea ingresar en el arreglo, y un arreglo de enteros que almacena los números ingresados por el usuario llamado **int[] numeros**

```
// Ciclo para llenar el arreglo con los elementos ingresados por el usuario
for (int i = 0; i < n; i++)
{
    Console.Write($"Elemento {i + 1}: ");
    numeros[i] = int.Parse(Console.ReadLine());
}
```

2. Se solicita al usuario que ingrese la cantidad de elementos que desea en el arreglo, luego, se crea un arreglo de enteros con la longitud especificada. Se utiliza un ciclo for para llenar el arreglo con los elementos ingresados por el usuario.

```
// Llama al método imprimirArreglo para mostrar el arreglo en orden original
funcionesYmetodos.imprimirArreglo(numeros);
Console.WriteLine();
```

3. Se llama al método **imprimirArreglo** para mostrar el arreglo en orden original.

```
// Llama al método imprimirArregloInvertido para mostrar el arreglo en orden invertido
funcionesYmetodos.imprimirArregloInvertido(numeros);
Console.ReadKey();
```

4. Se llama al método **imprimirArregloInvertido** para mostrar el arreglo en orden invertido.

```
// Llama al nuevo método para contar e imprimir números impares
funcionesYmetodos.ContarEImprimirImpares(numeros);
Console.ReadKey();
```

5. Se llama al método **ContarEImprimirImpares** para mostrar los números impares presentes en el arreglo y sus cantidad total.

**Clase de Funciones**
Esta clase contiene dos métodos estáticos para manipular y mostrar arreglos.

**1. imprimirArreglo(int[] arreglo)**

```
public static void imprimirArreglo(int[] arreglo)
{
    Console.Write("\nOriginal: ");

    // Recorre cada número en el arreglo y lo imprime en la consola
    foreach (int numero in arreglo)
    {
        Console.Write(numero + " ");
    }
}
```

6. Este método imprime los elementos del arreglo en su orden original.
   **Parámetros**
   **int[] arreglo: Arreglo de enteros que se desea imprimir.**

**2. imprimirArregloInvertido(int[] arreglo)**

```
public static void imprimirArregloInvertido(int[] arreglo)
{
    // Variable auxiliar para el intercambio de elementos
    int auxiliar;

    // Intercambia los elementos del arreglo para invertirlo
    for (int i = 0; i < arreglo.Length / 2; i++)
    {
        auxiliar = arreglo[i]; // Guarda el valor del elemento actual en la variable auxiliar
        arreglo[i] = arreglo[arreglo.Length - 1 - i]; // Asigna el elemento opuesto a la posición actual
        arreglo[arreglo.Length - 1 - i] = auxiliar; // Asigna el valor almacenado en auxiliar a la posición opuesta
    }

    // Recorre el arreglo invertido y lo imprime en la consola
    Console.Write("Inverso: ");
    for (int i = 0; i < arreglo.Length; i++)
    {
        Console.Write(arreglo[i] + " ");
    }
}
```

7. Este método invierte el arreglo y lo imprime.
   Utiliza un bucle para intercambiar elementos del arreglo de forma que el primer elemento se convierta en el último, el segundo en el penúltimo, y así sucesivamente. Luego imprime "Inverso: " seguido de los elementos del arreglo invertido, separados por espacios.
   **Parámetros**
   **int[] arreglo: Arreglo de enteros que se desea invertir e imprimir.**

```
public static void ContarEImprimirImpares(int[] arreglo)
{
    int contadorImpares = 0;

    Console.Write("\nNúmeros impares: ");

    // Recorre el arreglo y cuenta los números impares
    foreach (int numero in arreglo)
    {
        if (numero % 2 != 0) // Verifica si el número es impar
        {
            Console.Write(numero + " ");
            contadorImpares++;
        }
    }

    // Imprime la cantidad de números impares
    Console.WriteLine($"\nCantidad de números impares: {contadorImpares}");
}
```

8. Este método calcula la cantidad de números impares ingresados en un arreglo y los imprime. Primero recorre un arreglo de enteros, identifica los números impares presentes realizando una división por 2 para garantizar que el residuo sea 0, luego imprime los valores que coincidan con la condición y también muestra la cantidad total de números impares encontrados.
   **Parámetros**
   **int[] arreglo: Arreglo de enteros que se desea invertir e imprimir.**

**Ejercicio 1 (guía didáctica):** Crea un programa que solicite al usuario la base y la altura de un triángulo. El programa debe incluir una función llamada calcular_area_triangulo que reciba como parámetros la base y la altura, y devuelva el área del triángulo. La fórmula para calcular el área es: Área =(base * altura) /2

```
2 references
internal class AreaTriangulo
{
    // Método que recibe la base y la altura, y devuelve el área del triángulo
    1 reference
    public double calcular_area_triangulo(double baseTriangulo, double alturaTriangulo)
    {
        // La fórmula del área del triángulo: (base * altura) / 2
        return (baseTriangulo * alturaTriangulo) / 2;
    }
```

1. Se creó una clase externa al main llamada **AreaTriangulo**, que se utilizará para crear el método que calcule el área de un triángulo.
2. Dentro de la clase **AreaTriangulo**, se definió un método llamado **calcular_area_triangulo**, que recibe dos parámetros: **baseTriangulo** y **alturaTriangulo**, el cual va a calcular el área del triángulo utilizando estos valores.
3. En el método, se aplica la fórmula del área del triángulo, que es Área =(base * altura) /2, el resultado de esto se devuelve como un valor **double**, representando el área del triángulo basado en las dimensiones proporcionadas.

```
    // Solicitar la base del triángulo al usuario
    Console.WriteLine("Ingrese la base del triángulo:");
    double baseTriangulo = double.Parse(Console.ReadLine());

    // Solicitar la altura del triángulo al usuario
    Console.WriteLine("Ingrese la altura del triángulo:");
    double alturaTriangulo = double.Parse(Console.ReadLine());

    // Crear una instancia de la clase Triangulo para usar su método
    AreaTriangulo triangulo = new AreaTriangulo();

    // Llamar al método que calcula el área y guardar el resultado en la variable 'area'
    double area = triangulo.calcular_area_triangulo(baseTriangulo, alturaTriangulo);

    // Mostrar el área del triángulo en la consola
    Console.WriteLine($"El área del triángulo es: {area:F2}");

    Console.ReadKey();
}
```

4. Se piden al usuario la base y la altura del triángulo. Los valores ingresados se convierten a tipo double utilizando double.Parse para poder realizar cálculos precisos.
5. Se crea una instancia de la clase **AreaTriangulo**, llamada **triangulo**. Esto permite acceder a los métodos de la clase para calcular el área del triángulo.
6. Se llama al método **calcular_area_triangulo** de la instancia **triangulo**, pasando la base y la altura como argumentos. El resultado se guarda en la variable area, y luego se muestra en la consola con formato de dos decimales usando **$"{area:F2}",** que presenta el área calculada de manera legible.