UNIVERSIDAD AMERICANA

Facultad de Ingeniería y Arquitectura

Metodología y programación estructurada

Paso de Arreglos a funciones

Estudiantes:

- Diego Armando Urbina Avilés
- Emmanuel Leonardo Aguilar Novoa

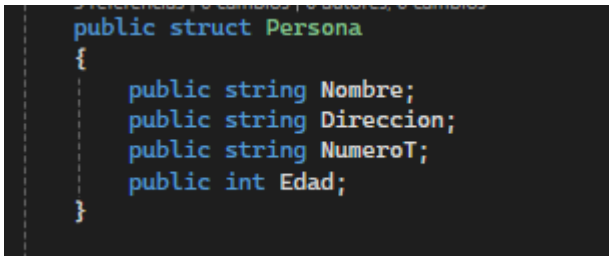
Docente:

Silvia Gigdalia Ticay Lopez

Programa #1, Guía de Evaluación

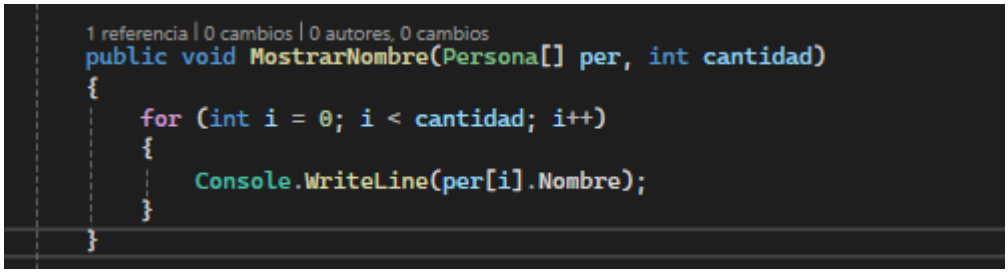
1. Programa que sea capaz de almacenar los datos de 10 personas: nombre, dirección, teléfono, edad (usando structs). Deberá ir pidiendo los datos uno por uno, hasta que el usuario lo decida. Entonces deberá aparecer un menú que permita:

- 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.



```
public struct Persona
{
    public string Nombre;
    public string Direccion;
    public string NumeroT;
    public int Edad;
}
```

Primeramente, definimos el struct que almacenara los datos que nos pide el problema, siendo estos nombre, direcciones, número de teléfono y la edad.

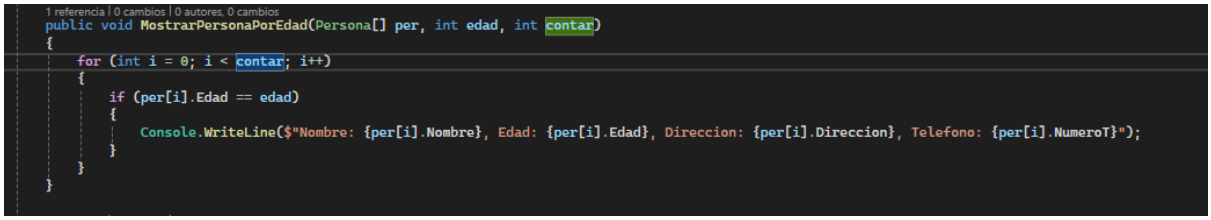


```
1 referencia | 0 cambios | 0 autores, 0 cambios
public void MostrarNombre(Persona[] per, int cantidad)
{
    for (int i = 0; i < cantidad; i++)
    {
        Console.WriteLine(per[i].Nombre);
    }
}
```

Luego tenemos la función “MostrarNombre”, el método Persona[ ] recibe como parámetro un arreglo de tipo estructura en este caso “Persona”, a este lo llamamos “per”, y cantidad representa la cantidad de nombres de personas que se quieren mostrar.

El bucle for va a iterar sobre el arreglo, este va a continuar mientras “i” sea menor que el numero de personas de las cuales se quiere mostrar su nombre, y después de cada iteración que este realice el valor de “i” se incrementa en 1.

Luego se imprime el nombre de la persona, se accede a la persona que se encuentra en la posición “i” del arreglo “per[ ]”, y .Nombre que representa el valor que se almacena en el estruct.

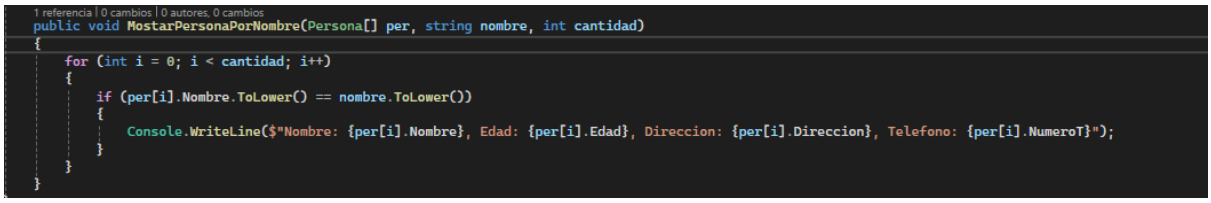


```
1 referencia | 0 cambios | 0 autores, 0 cambios
public void MostrarPersonaPorEdad(Persona[] per, int edad, int contar)
{
    for (int i = 0; i < contar; i++)
    {
        if (per[i].Edad == edad)
        {
            Console.WriteLine($"Nombre: {per[i].Nombre}, Edad: {per[i].Edad}, Direccion: {per[i].Direccion}, Telefono: {per[i].NumeroT}");
        }
    }
}
```

Al igual que en la función anterior “MostrarpersonaPorEdad”, el método “Persona [ ] per” es un arreglo, luego declaramos dos variables de tipo entero, una para almacenar la edad que se desea buscar y otra que es para el numero total de las personas en el arreglo.

El bucle “for” va recorrer todos lo elementos del arreglo hasta el número que será especificado por la variable “contar”, se añade un condicional simple, que va a verificar si la Edad de la persona que se encuentra en la posición “i” es igual a la edad que se pasa como parámetro, y per[i] accede a “Edad” del estruct Persona en la posición “i”.

Luego si la condición se cumple, si la persona tiene la edad buscada, se imprimen todos los datos de esta, nombre, teléfono etc.



```
1 referencia | 0 cambios | 0 autores, 0 cambios
public void MostarPersonaPorNombre(Persona[] per, string nombre, int cantidad)
{
    for (int i = 0; i < cantidad; i++)
    {
        if (per[i].Nombre.ToLower() == nombre.ToLower())
        {
            Console.WriteLine($"Nombre: {per[i].Nombre}, Edad: {per[i].Edad}, Direccion: {per[i].Direccion}, Telefono: {per[i].NumeroT}");
        }
    }
}
```

La función “MostrarPersonaPorNombre” es similar a la anterior “MostrarPersonaPorEdad” la diferencia es en el condicional al momento de realizar la verificación, en esta funcion el condicional, se compara el nombre de la persona además de que se utiliza la funcion ToLower() para que la comparación entre

mayúsculas y minúsculas sea indiferente, luego de que se cumpa la condición se imprimen todos los datos de la persona buscada.

```
AdministrarPer.Persona[] persona = new AdministrarPer.Persona[10]; // Arreglo para almacenar los datos de las personas
AdministrarPer ADPersona = new AdministrarPer(); // Instancia de la clase PersonManager
int AC = 0;
bool iniciar = true;
```

AdministrarPer.Persona[] persona = new AdministrarPer.Persona[10], esta línea crea un arreglo para almacenar los 10 struct “Persona” que se encuentran en la clase AdministrarPer. AdministrarPer ADPersona = new AdministrarPer(), aquí se crea una nueva instancia de la clase “AdministrarPer” y se guarda en la variabla “ADPersona”. Luego inicializamos una variable que utilizaremos como contador y la variable “iniciar” como true para iniciar el ciclo “while” para poder desplegar un menú.

```
while (iniciar)
{
    Console.Clear();
    //  menú de opciones
    Console.WriteLine("1. Añadir una persona");
    Console.WriteLine("2. Mostrar todos los nombres");
    Console.WriteLine("3. Mostrar persona por edad");
    Console.WriteLine("4. Mostrar persona por nombre");
    Console.WriteLine("5. Salir del programa");
    Console.Write("Eliga una opcion: ");
    string OP = Console.ReadLine();
```

Este es el menú que le permite al usuario añadir una persona, mostrar todos los nombres almacenados y buscar una persona por su nombre o edad, la variable OP de tipo “string” se utilizara para el condicional “switch” y poder acceder a los casos deseados.

```
switch (OP)
{
    case "1":
        if (AC < 10) // Limite de 10 personas
        {
            // solicitud de datos de la persona
            Console.Write("Ingrese un nombre: ");
            persona[AC].Nombre = Console.ReadLine();
            Console.Write("Ingrese una direccion: ");
            persona[AC].Direccion = Console.ReadLine();
            Console.Write("Ingrese el numero de telefono: ");
            persona[AC].NumeroT = Console.ReadLine();
            Console.Write("Ingrese la edad: ");
            persona[AC].Edad = int.Parse(Console.ReadLine());
            AC++;
        }
        else
        {
            Console.WriteLine("Se alcanzo el limite de personas.");
        }
        break;

    case "2":
        ADPersona.MostrarNombre(persona, AC);
        break;

    case "3":
        Console.Write("Ingrese la edad para buscar a la persona: ");
        int edad = int.Parse(Console.ReadLine());
        ADPersona.MostrarPersonaPorEdad(persona, edad, AC);
        break;

    case "4":
        Console.Write("Ingre el nombre de la persona a buscar: ");
        string nombre = Console.ReadLine();
        ADPersona.MostarPersonaPorNombre(persona, nombre, AC);
        break;

    case "5":
        iniciar = false;
        break;

    default:
        Console.WriteLine("Opcion no valida.");
        break;
}
Console.ReadKey();
```

Simplemente en cada caso se hace el llamado de la función correspondiente en base a que realizara cada opción del menú.

Programa #5, Guía de Evaluación

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.

```
2 referencias | 0 cambios | 0 autores, 0 cambios
public static void MostrarArreglo(int[] arreglo)
{
    for (int i = 0; i < arreglo.Length; i++)
    {
        Console.Write(arreglo[i] + " ");
    }
    Console.WriteLine();
}
```

La funcion “MostrarArrrglo” toma un arreglo de tipo entero como parámetro y lo muestra al usuario, utiliza un ciclo “for” para recorrer todos los elementos de este arreglo, y en cada itereacion que realiza imprime el valor actual que tiene el arreglo.

```
1 referencia | 0 cambios | 0 autores, 0 cambios
public static int[] ArregloInvertido(int[] arreglo)
{
    int[] invertido = new int[arreglo.Length];
    for (int i = 0; i < arreglo.Length; i++)
    {
        invertido[i] = arreglo[arreglo.Length - 1 - i];
    }
    return invertido;
}
```

Esta función “ArregloInvertido” va a devolver un nuevo arreglo, el cual va a contener los mismo elementos que el arreglo original pero los devuelve de forma inversa, primero crea un arreglo llamado invertido que tiene el mismo tamaños que el arreglo original, utiliza un ciclo “for” que recorre los elementos del arreglo original y en cada iteración que realiza se le asigna el valor del elemento que se encuentra en la posición “i” del arreglo original a la posición invertida que corresponde en el arreglo invertido mediante “arreglo.Lenght -1 -i” y al final devuelve el nuevo arreglo ya con los elementos en orden inverso.

```
1 referencia | 0 cambios | 0 autores, 0 cambios
public static int NumeroDeImpares(int[] arreglo)
{
    int cont = 0;
    for (int i = 0; i < arreglo.Length; i++)
    {
        if (arreglo[i] % 2 != 0)
        {
            cont++;
        }
    }
    return cont;
}
```

Y la funciones “NumeroDeImpares” se utiliza para contar la cantidad números impares que se encuentran en el arreglo, se declara una variable “count” que se inicializa en 0, el ciclo “for” va a recorrer todos los elementos del arreglo, y en cada iteración, si el valor del arreglo es impar que se verifica mediante “arreglo[i] % 2 != 0” primero se toma el elemento que se encuentra en la posición “i” del arreglo y lo divide entre 2 mediante el operador de modulo “%” devolviendo la división de esto, si esto es diferente de 0, significa que el numero es impar.

```
0 referencias | 0 cambios | 0 autores, 0 cambios
static void Main(string[] args)
{
    //Creamos un arreglo para ejemplificar el caso
    int[] arregloOriginal = { 1, 2, 3, 4, 5 };

    Console.WriteLine("Arreglo original:");
    Funciones.MostrarArreglo(arregloOriginal);

    Console.WriteLine("Arreglo invertido:");
    int[] arregloInvertido = Funciones.ArregloInvertido(arregloOriginal);
    Funciones.MostrarArreglo(arregloInvertido);

    int impares = Funciones.NumeroDeImpares(arregloInvertido);
    Console.WriteLine($"La cantidad de numeros impares en el arreglo invertido es: {impares}");
}
```

Ya simplemente declaramos nuestro arreglo original y llamamos a todos las funciones previamente declaradas.

Programa #3, Guía de Evaluación

3. Crea un programa que use una lista genérica para almacenar temperaturas en grados Celsius. Implementa funciones para convertirlas a Fahrenheit y Kelvin, y mostrar las temperaturas convertidas, las cuales se almacenan en nuevas listas respectivamente.

- El programa se repetirá las veces que el usuario lo decida.
- Utilizar funciones con parámetros.
- Agregar una función que permita eliminar de la lista que contiene las temperaturas convertidas. La que el usuario elija

```
public static double ConvertirAFahrenheit(double celsius)
{
    return (celsius * 9 / 5) + 32;
}
```

Esta función recibe un parámetro de tipo double al cual llamamos Celsius, esto representa la temperatura en grados Celsius, luego recibe el parámetro de Celsius lo multiplica por 9, lo divide entre 5 y le suma 32, esto es la formula que se utiliza para convertir una temperatura de grados Celsius a Fahrenheit.

```
1 referencia | 0 cambios | 0 autores, 0 cambios
public static double ConvertirAKelvin(double celsius)
{
    return celsius + 273.15;
}
```

Esta función realiza prácticamente los mismo que la anterior, recibe un parámetro double llamado Celsius, luego a este se le suma 273.15 y regresa este valor que es la temperatura en Kelvin.

```
2 referencias | 0 cambios | 0 autores, 0 cambios
public static void ImprimirListaTemperaturas(List<double> temperaturas, string tipo)
{
    Console.WriteLine($"Temperaturas en {tipo}:");
    foreach (var temp in temperaturas)
    {
        Console.WriteLine($"{temp:F2}");
    }
}
```

List<double> temperaturas: Es el primer parámetro de la función, que es una lista de números decimales (double), representando las temperaturas.

string tipo: El segundo parámetro es un string que representa el tipo de temperatura, "Celsius" o "Fahrenheit"). Console.WriteLine($"Temperaturas en {tipo}, esta línea imprime el encabezado que indica el tipo de temperatura. Utiliza interpolación de cadenas (la expresión dentro de las llaves {} para incluir el valor de tipo en la salida.“foreach” (var temp in temperaturas), este es un bucle “foreach” que recorre cada valor (temp) en la lista temperaturas. “var temp” significa que el tipo de temp es inferido, en este caso será double porque la lista es de tipo List<double>.

Console.WriteLine($"{temp:F2}") cada valor de “temp” se imprime en la consola con formato de dos decimales (F2 nos indica que el número debe ser mostrado con dos decimales.

```
public static void EliminarTemperatura(List<double> temperaturas)
{
    Console.WriteLine("Indica la posicion de la temperatura que deseas eliminar (empieza desde 0):");
    int indice = int.Parse(Console.ReadLine());

    if (indice >= 0 && indice < temperaturas.Count)
    {
        temperaturas.RemoveAt(indice);
        Console.WriteLine("Temperatura eliminada");
    }
    else
    {
        Console.WriteLine("Invalido");
    }
```

List<double> temperaturas: Es el parámetro que recibe la lista de temperaturas de tipo double, luego se nos indica la posición de la temperatura que deseamos eliminar, y que esta empieza desde 0, luego mediante el condicional “if” verificamos si el índice que se ha ingresado es válido, es decir que esta dentro del rango de la lista, que es mayor o igual a 0 y menor que el numero de los elementos de la lista, si el índice es valido “RemoveAt” elimina la temperatura en esta posición, y si este índice no se encuentra dentro de los limites se imprime un mensaje de “Invalido”.

```
List<double> temperaturasCelsius = new List<double>();
List<double> temperaturasFahrenheit = new List<double>();
List<double> temperaturasKelvin = new List<double>();

bool continuar = true;

while (continuar)
{
    Console.WriteLine("Indica la temperatura en grados Celsius:");
    double temperaturaCelsius = double.Parse(Console.ReadLine());
    temperaturasCelsius.Add(temperaturaCelsius);

    // Convertir a Fahrenheit y Kelvin
    double fahrenheit = ConversionTemperaturas.ConvertirAFahrenheit(temperaturaCelsius);
    double kelvin = ConversionTemperaturas.ConvertirAKelvin(temperaturaCelsius);

    temperaturasFahrenheit.Add(fahrenheit);
    temperaturasKelvin.Add(kelvin);

    // Mostrar las listas de temperaturas convertidas
    ConversionTemperaturas.ImprimirListaTemperaturas(temperaturasFahrenheit, "Fahrenheit");
    ConversionTemperaturas.ImprimirListaTemperaturas(temperaturasKelvin, "Kelvin");

    // Preguntar si desea eliminar una temperatura
    Console.WriteLine("¿Deseas eliminar una temperatura de las listas? (s/n)");
    string respuestaEliminar = Console.ReadLine().ToLower();

    if (respuestaEliminar == "s")
    {
        Console.WriteLine("¿De cual lista deseas eliminar? (1. Fahrenheit, 2. Kelvin)");
        int opcionLista = int.Parse(Console.ReadLine());

        if (opcionLista == 1)
        {
            ConversionTemperaturas.EliminarTemperatura(temperaturasFahrenheit);
        }
        else if (opcionLista == 2)
        {
            ConversionTemperaturas.EliminarTemperatura(temperaturasKelvin);
        }
        else
        {
            Console.WriteLine("Opcion no valida.");
        }
    }

    // Preguntar si desea continuar
    Console.WriteLine("¿Deseas ingresar otra temperatura? (s/n)");
    string respuesta = Console.ReadLine().ToLower();
    if (respuesta != "s")
    {
        continuar = false;
    }
```

En el main inicializamos las listas para almacenar las temperaturas , y utilizamos un ciclo “while” para que se le permita al usario continuar añadiendo temperaturas, y mediante condicionales anidados se le muestran las opciones de eliminar temperaturas al usuario.

Ejercicio #3, Guía Didáctica

Ejercicio No. 3 Desarrolla un programa que gestione las calificaciones de un grupo de estudiantes. El programa debe:

- ✓ Permitir ingresar las calificaciones de varios estudiantes.
- ✓ Calcular el promedio de calificaciones de cada estudiante.
- ✓ Determinar el estudiante con el promedio más alto y el más bajo.

Requisitos:

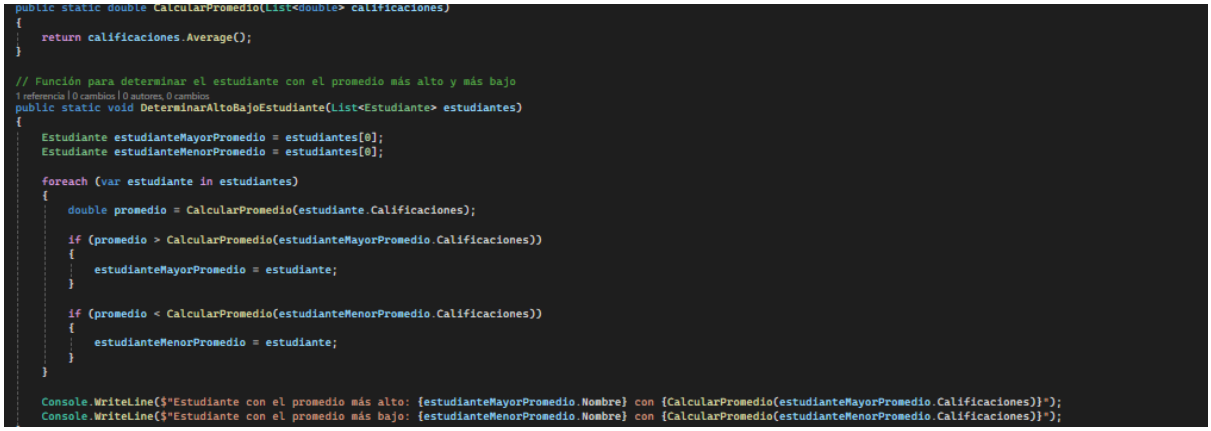
- ✓ Implementar la función agregar_estudiante (estudiantes, nombre, calificaciones) para agregar estudiantes y sus calificaciones (una lista de notas).
- ✓ Implementar la función calcular_promedio (calificaciones) para calcular el promedio de un estudiante.
- ✓ Implementar la función determinar_alto_bajo_estudiante (estudiantes) que devuelva los nombres del estudiante con el promedio más alto y el promedio más bajo.

```
public struct Estudiante
{
    public string Nombre;
    public List<double> Calificaciones;
}
```

Primero creamos la estruct para poder registrar los nombre y calificaciones de los estudiantes.

```
public static void AgregarEstudiante(List<Estudiante> estudiantes, string nombre, List<double> calificaciones)
{
    Estudiante nuevoEstudiante = new Estudiante
    {
        Nombre = nombre,
        Calificaciones = calificaciones
    };
    estudiantes.Add(nuevoEstudiante);
}
```

Declaramos el primer método al cual llamamos “agregarestudiante” este es un método static, por lo que se puede llamar sin necesidad de instanciar un objeto de la clase que contenga a este, este recibe 3 parámetros, que son una lista de objetos a la que se le agrega un nuevo estudiante “List<Estudiante> estudiantes”, una cadena de caracteres para almacenar los nombre de los estudiantes que se desean agregar, y otra lista de tipo double para las calificaciones asociadas a cada estudiante “List<double> calificaciones”.



```
public static double CalcularPromedio(List<double> calificaciones)
{
    return calificaciones.Average();
}

// Función para determinar el estudiante con el promedio más alto y más bajo
1 referencia | 0 cambios | 0 autores, 0 cambios
public static void DeterminarAltoBajoEstudiante(List<Estudiante> estudiantes)
{
    Estudiante estudianteMayorPromedio = estudiantes[0];
    Estudiante estudianteMenorPromedio = estudiantes[0];

    foreach (var estudiante in estudiantes)
    {
        double promedio = CalcularPromedio(estudiante.Calificaciones);

        if (promedio > CalcularPromedio(estudianteMayorPromedio.Calificaciones))
        {
            estudianteMayorPromedio = estudiante;
        }

        if (promedio < CalcularPromedio(estudianteMenorPromedio.Calificaciones))
        {
            estudianteMenorPromedio = estudiante;
        }
    }

    Console.WriteLine($"Estudiante con el promedio más alto: {estudianteMayorPromedio.Nombre} con {CalcularPromedio(estudianteMayorPromedio.Calificaciones)}");
    Console.WriteLine($"Estudiante con el promedio más bajo: {estudianteMenorPromedio.Nombre} con {CalcularPromedio(estudianteMenorPromedio.Calificaciones)}");
```

Esta funcion recibe una lista de calificaciones y devuelve el promedio de estas mediante el método .Average(), este cacula la media de los valores de una lista.

Luego la funcion para determinar el promedio mas alto y mas bajo “DeterminarAltoBajoEstudiante” recibe una lista de estudiantes e inicializa 2 variables, que son “estudianteMayorPromedio” y “estudianteMenorPromedio” inicialmente se asignan al primer estudiante de nuestra lista.

Luego el bucle Foreach compara los promedios , recorre cada estudiante en la lista, y por cada estudiante se calcula el promedio mediante la funcion “CalcularPromedio” luego se comparan los promedios mediante condicionales, para actualizar los valores e imprimirlos en pantalla.