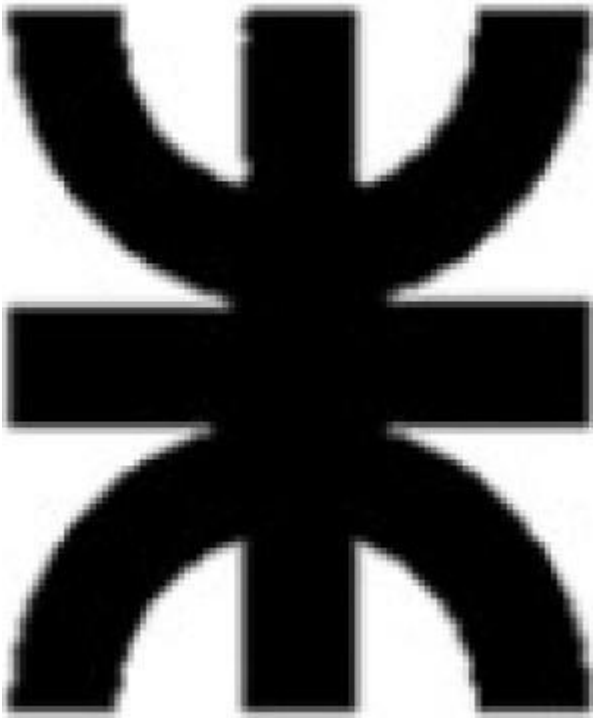


UNIVERSIDAD TECNOLÓGICA NACIONAL
Facultad Regional Reconquista

Programación en Computación

Ciclo Lectivo: 2020

Trabajo Práctico N.º 7

7 – Estructuras REPETITIVAS

GRUPO N.º: 12
INTEGRANTES: Fulano, Mengano, Zutano

Guía Unidad 7: ALGORITMOS REPETITIVOS

SITUACIONES PROBLEMÁTICAS:

1. **Ingresar N números. Imprimir cantidad de positivos, cantidad de negativos y porcentaje de ceros sobre el total de números ingresados.**
2. **Ingresar N números de a uno por vez. Imprimir el promedio de pares y porcentaje de impares sobre la cantidad de números ingresados.**
3. **Ingresar N números y obtener:**
   - 3.1. Porcentaje de pares sobre la cantidad de números ingresados.
   - 3.2. Cantidad de números mayores a 30
4. **Se ingresan las notas de 10 alumnos. Imprimir:**
   - 4.1. Promedio general de notas.
   - 4.2. Porcentaje de alumnos aprobados. (Se aprueba con 6.)
   - 4.3. Cantidad de alumnos no aprobados.
5. **Suponga que se tiene un conjunto de calificaciones de un grupo de N alumnos. Realizar un algoritmo para calcular la nota promedio la calificación más baja, la nota máxima y el orden en que se ingresó** (número de alumno)**.**
6. **Calcular e imprimir la tabla de multiplicar de un número cualquiera. Imprimir el multiplicando, el multiplicador y el producto.**
7. **Se ingresan las notas de N alumnos (notas de 1 a 10). Imprimir:**
   - 7.1. Cantidad de alumnos aprobados. (Se aprueba con 6.)
   - 7.2. Cantidad de alumnos desaprobados.
   - 7.3. Cantidad de alumnos con notas superior a 8.
   - 7.4. Porcentaje de alumnos aprobados.
8. **Ingresando 10 valores numéricos determinar:**
   - 8.1. Cantidad de valores numéricos que son menores a 100 o mayores a 500.
   - 8.2. Suma de los valores estén comprendidos entre 200 y 300
9. **Se solicitan las notas de un alumno. Obtener el promedio, la nota más alta, la nota más baja. No se conoce la cantidad de notas.**
10. **Ingresar al comienzo de un programa dos datos numéricos A, B.** (Teniendo en cuenta que A debe ser menor a B). **Luego ingresar 10 datos más, de a uno por vez e imprimir los datos:**
    - 10.1. Que se encuentran comprendidos entre A y B, así como su suma y promedio.
    - 10.2. Que NO se encuentran comprendidos entre A y B, así como su suma y promedio.
11. **Un objeto es dejado caer desde una altura de 100 m. Diseñe un programa que imprima cada décima de segundo la distancia entre el objeto y el suelo. Al final, imprima el tiempo necesario en décimas de segundo para que el objeto toque el suelo.**
12. **Calcular la suma siguiente: 100 + 98 + 96 + 94 + . . . + 0 en este orden.**

ACTIVIDADES:

Resolver las situaciones problemáticas anteriores, debidamente comentadas y comenzando con:

1. **ANALIZAR** el Problema, *Datos de Entrada, Salida y Auxiliares, y ¿Que ocurre sí?...*
2. Realizar el **pseudocódigo**,
3. **Diagrama de Flujo** en RAPTOR,
4. Codificación en **VSC# Consola**.
5. Codificación en **VSC# Ventana**. (Windows Form)

CONOCIMIENTOS NECESARIOS:

U 2) Algoritmos. U 3) Tipos de Datos. U 4) Diagramación lógica. Diagramas de Flujo. U 5) Estructuras Secuenciales. Uso de Asignaciones, Entradas y Salidas de Datos. U 6) Estructuras de selección o condicionales. Uso de condicionales para la formulación de algoritmos. U 7) Estructuras Repetitivas o Cíclicas: HACER-PARA, HACER-MIENTRAS, REPETIR-HASTA. Resolución de problemas usando todos los tipos de estructuras para la formulación de sus algoritmos.

OPERATORIA:

Varia con cada situación problemática presentada, pero se resuelve en base a los conocimientos necesarios considerados.

RECOMENDACIÓN:

Para proceder a la realización de los TPs de esta Guía, es recomendable releer la siguiente bibliografía para un repaso de los conceptos teóricos involucrados en resolución de estos problemas:

Diseño_de_Algoritmos.pdf, En especial Capítulo 5.

Curso de Algoritmia.pdf, En especial Capítulo 5.

Problema 12:

Calcular la suma siguiente: 100 + 98 + 96 + 94 + . . . + 0 en este orden.

.

1) Análisis del Problema:

El problema nos pide obtener un número que es la suma de una serie ordenada descendente en 2 desde 100 a 0.
Como DATOS necesitamos:
CONSTANTE_INICIO = 100, como número de partida
CONSTANTE_PASO = -2, como número de decremento de la serie
Contador, Como Variable para realizar el conteo de los sumandos
Suma = 0, Como Variable para ir acumulando los valores a sumar.
El problema se resuelve utilizando una estructura de Ciclo (o Repetitiva) controlada por la variable ***Contador***, la cual se inicializa con el valor de ***CONSTANTE_INICIO,*** donde iremos descontando la ***CONSTANTE_PASO*** en cada Ciclo hasta alcanzar el valor de ***Contador*** = 0. Para obtener el resultado, a la variable ***Suma*** se le irá sumando el valor de ***Contador*** en cada ciclo.

2) Pseudo-Código:

Es un lenguaje muy libre, que utiliza **palabras reservadas** y exige las **sangrías.** La estructura básica es:

Algoritmo <TP7_G12_Ej12>

```
// Realizar la suma ordenada de la serie descendente de pares de 100 a 0.
// Calcular la suma siguiente:   100 + 98 + 96 + 94 + . . .+ 2 + 0
// Fecha de Entrega: DD/MM/AA.
// Programador: Grupo 12 de PeC.


Variables          // definiciones de las variables y de su tipo.
    Entrada:       // No se definen
    Salida:
        Texto txtMensaje ;  // Carga el Mensaje de Salida
    Auxiliares:
        Entero eContador, eSuma, eCONSTANTE = 100, ePASO = -2 ;
        // Usadas en el proceso de Calculo
```

```
INICIO          // Inicio del Pseudocódigo
        Llamar (PresentacionDelProblema);
        Llamar (Calculo);
        Llamar (MostrarResultado);
FIN             // Fin del Pseudocódigo

// Llamados a SubProgramas o Módulos o Funciones del programa

SubProgr(PresentacionDelProblema)

INICIO
        Imprimir ("Calcular el valor de la suma siguiente: ")
        Imprimir ("100 + 98 + 96 + 94 + ... + 10 + 8 + 6 + 4 + 2 + 0")
FIN

SubProgr(Calculo)

INICIO
        Asignar eContador ← eCONSTANTE, eSuma ← 0;
        Asignar txtMensaje ← "Suma = " ;
        HACER-MIENTRAS (eContador <> 0)
                Asignar eSuma ← eSuma + eContador ;
                Asignar txtMensaje ← txtMensaje + eContador + " + " ;
                Asignar eContador ← eContador + ePASO ;
        FIN-HACER-MIENTRAS
FIN

SubProgr(MostrarResultado)

INICIO
        Imprimir (txtMensaje + "0 = " + eSuma)
FIN
```

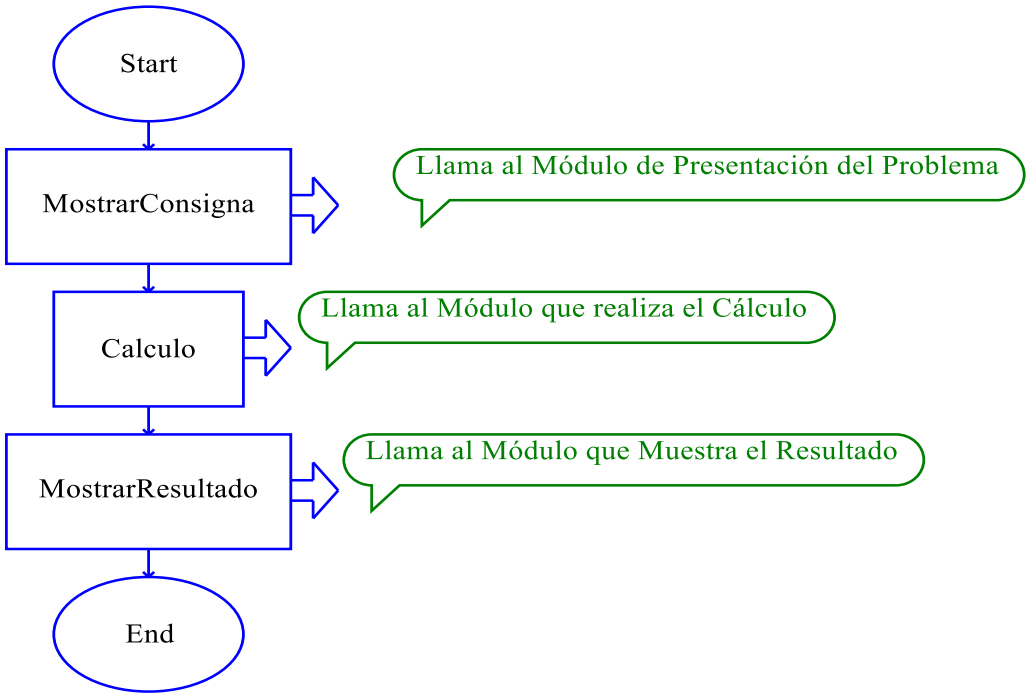
3) Diagramas de Flujo.

Utiliza **símbolos normalizados** unidos por flechas (**líneas de flujo**) que indican el orden en que debe ejecutarse cada paso.

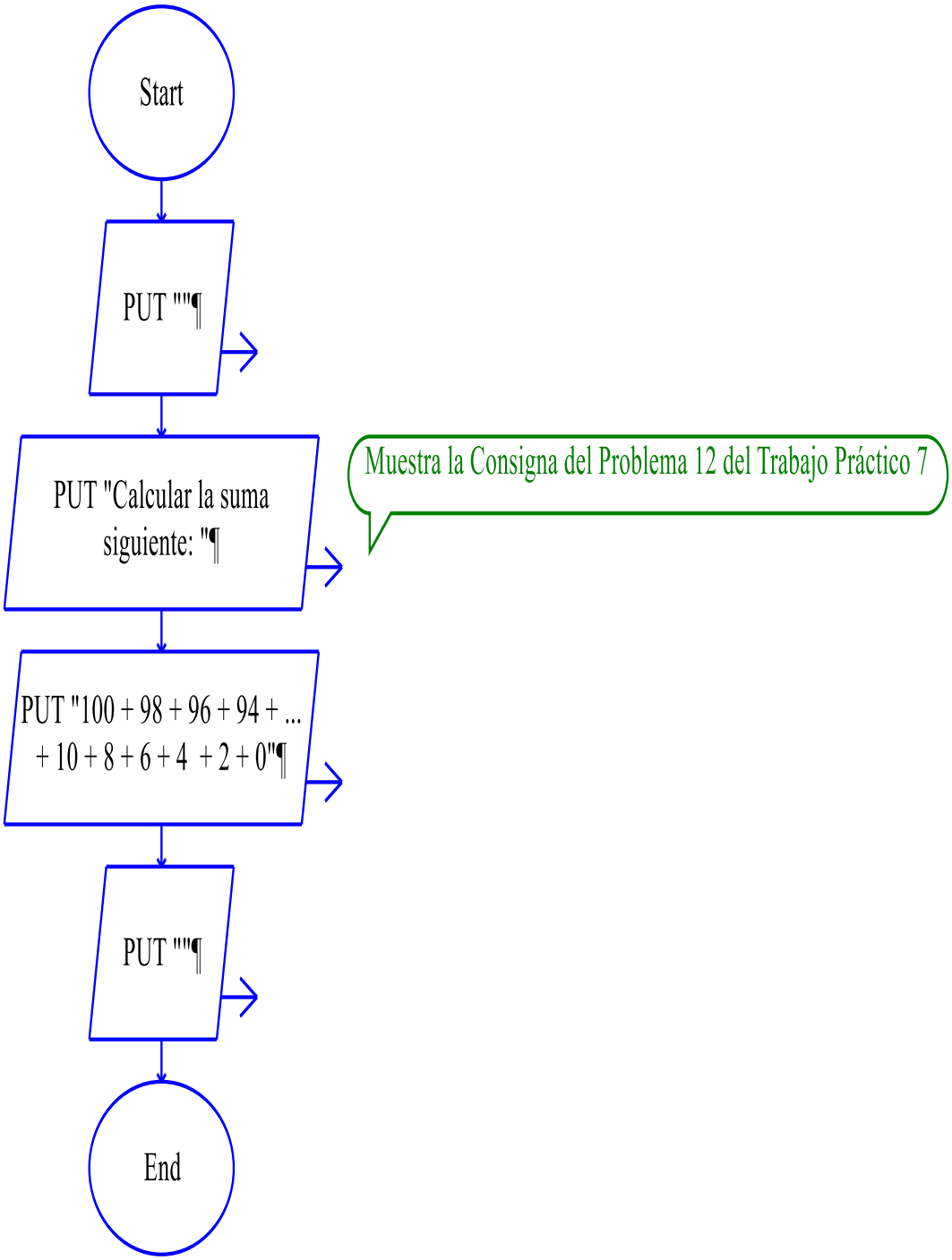
Es muy importante porque permite:

- Ver el flujo del Programa.
- Analizar la semántica del Condicional.
- Entender como piezas individuales interactúan para formar programas más complejos.

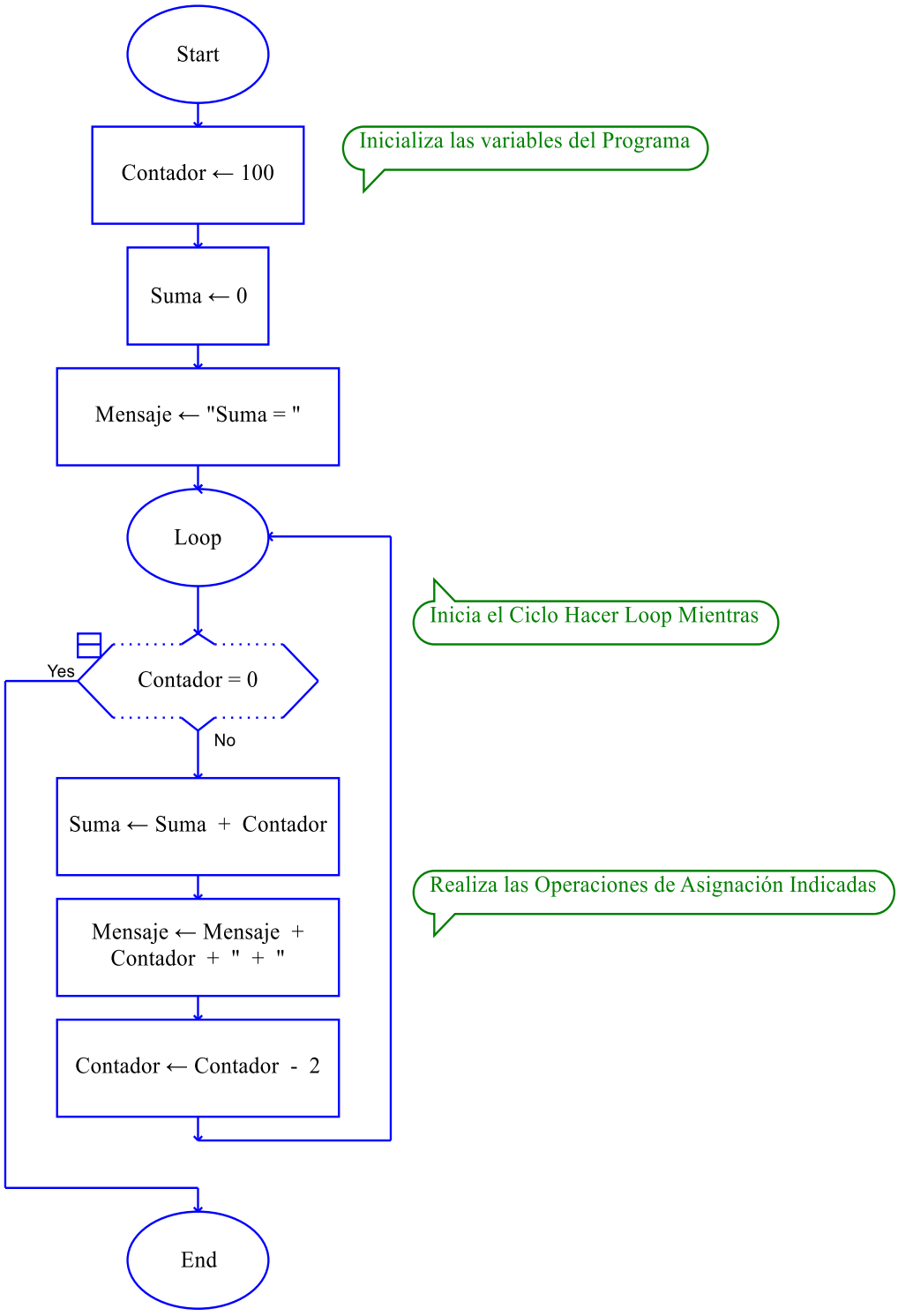
Programa **"main"**



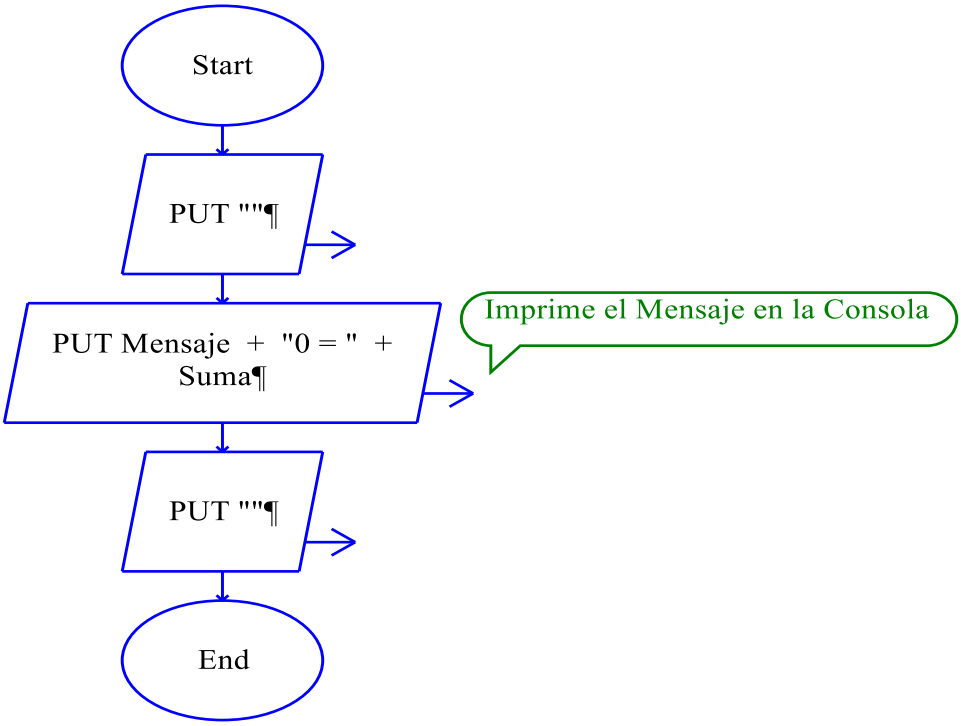
Subrutina **“MostrarConsigna”**



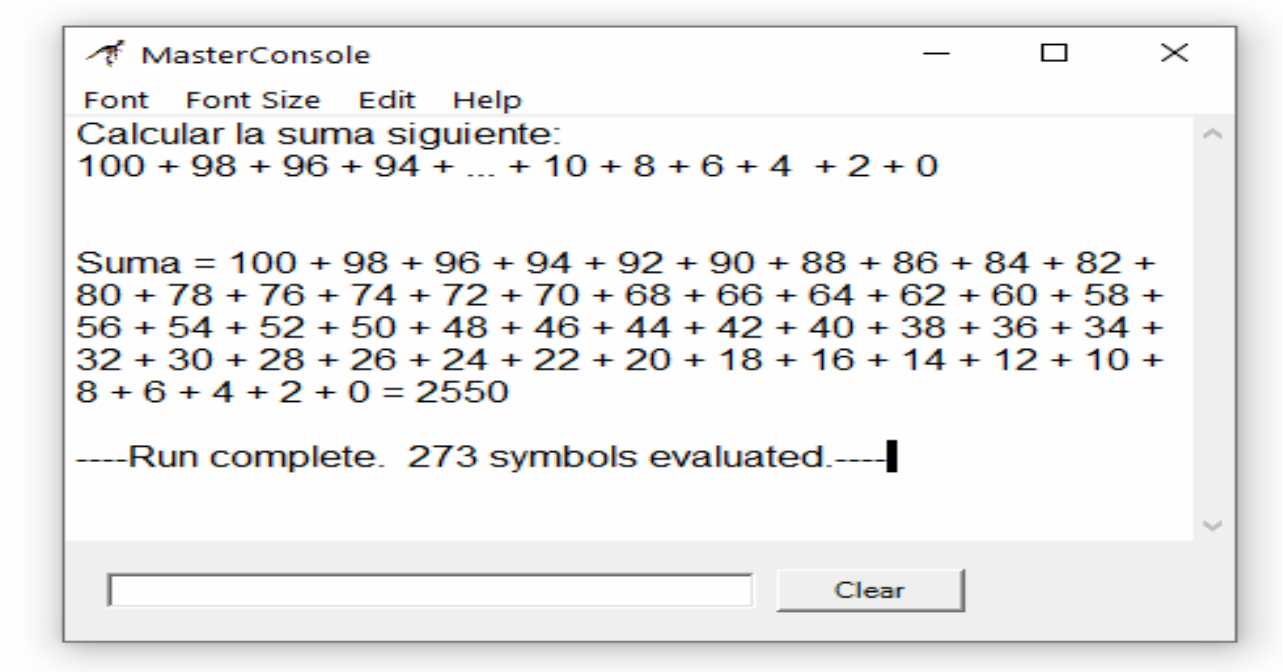
Subrutina **“Calculo”**



Subrutina **“MostrarResultado”**



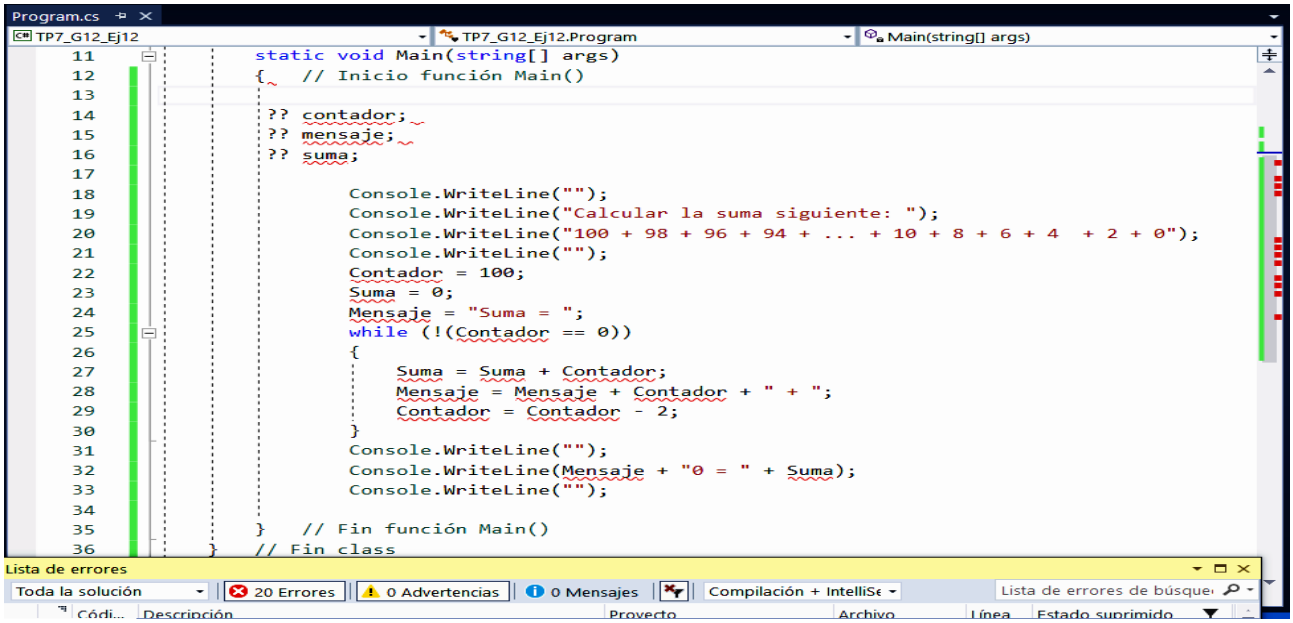
Realizado el DF en RAPTOR, lo ejecutamos y nos muestra en Consola:



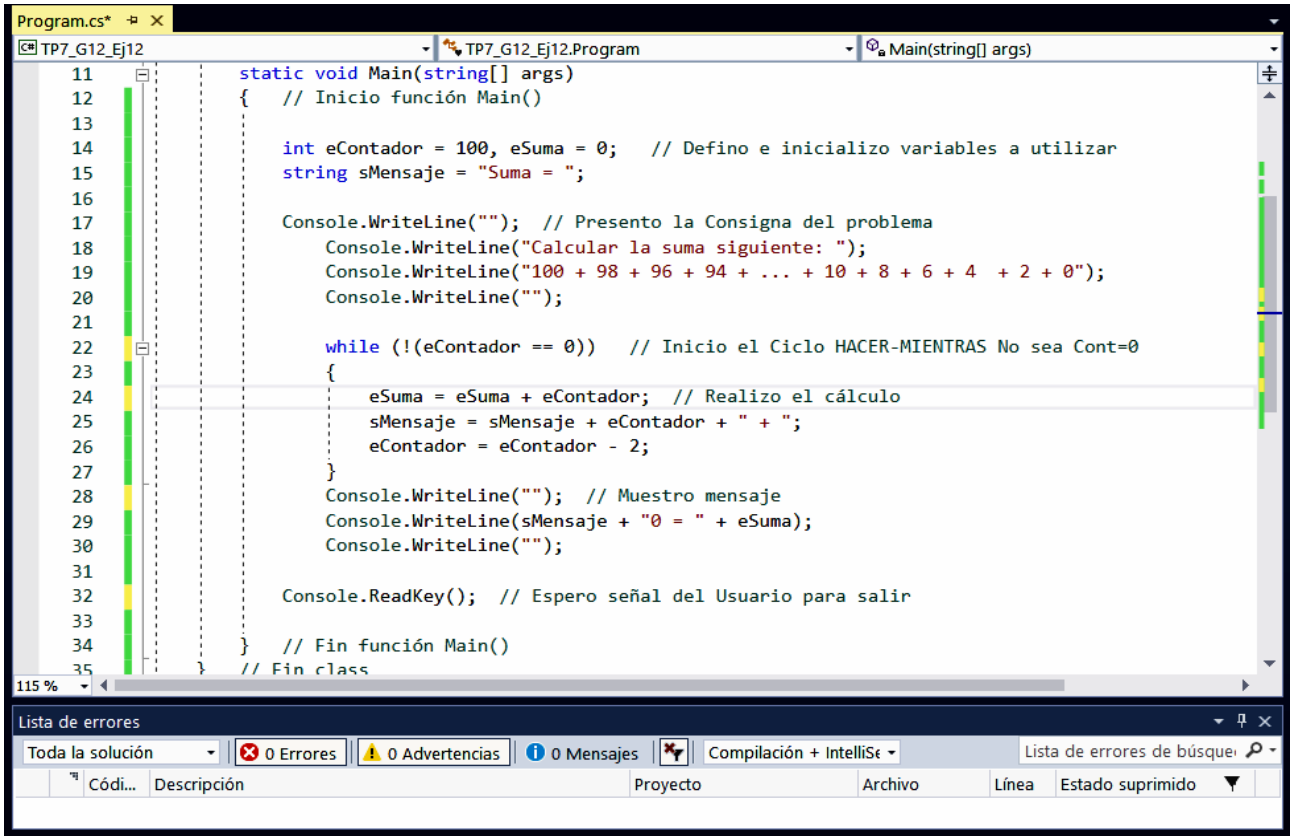
Analizar la construcción de la variable Mensaje para obtener la salida observada.

4.a) Codificación en C# (Consola).

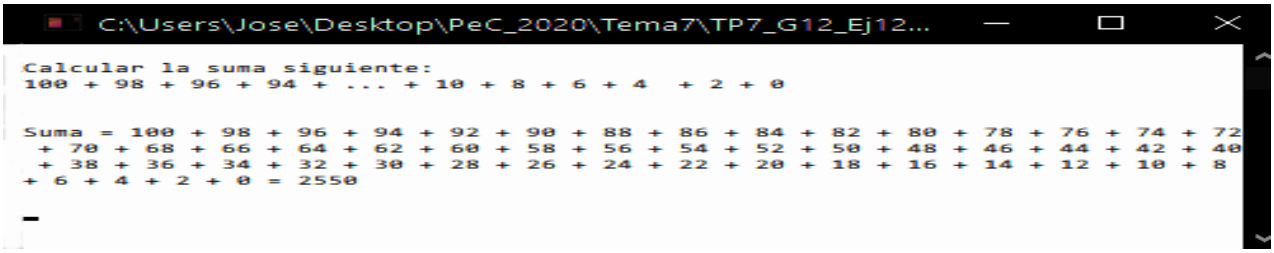
Vemos la copia del código C# generado por Raptor y cargado en Visual Studio C# en la imagen siguiente:



Como era de esperar, la cantidad de Errores es alta (20) y no tenemos Advertencias ni Mensajes, pero corrigiendo solo la definición de las variables, ya estamos en 0 Errores



Programa ejecutándose:



El programa se está ejecutando en el modo consola del S.O. Windows

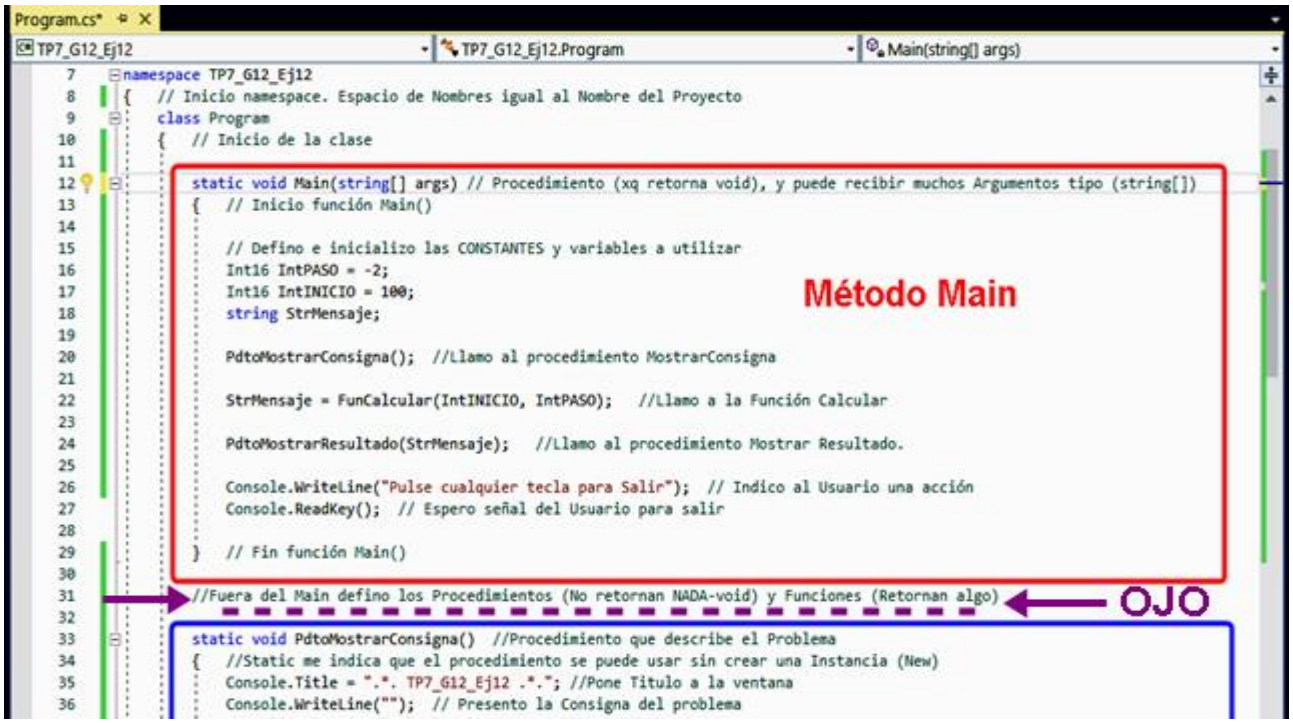
Quiero recordar que, dejando de lado las correcciones realizadas en la definición de variables, la lógica o algoritmo del programa en Visual C# es el que hemos Generado en Raptor para C#.

4.b) Codificación en C# con Subrutinas (Consola).

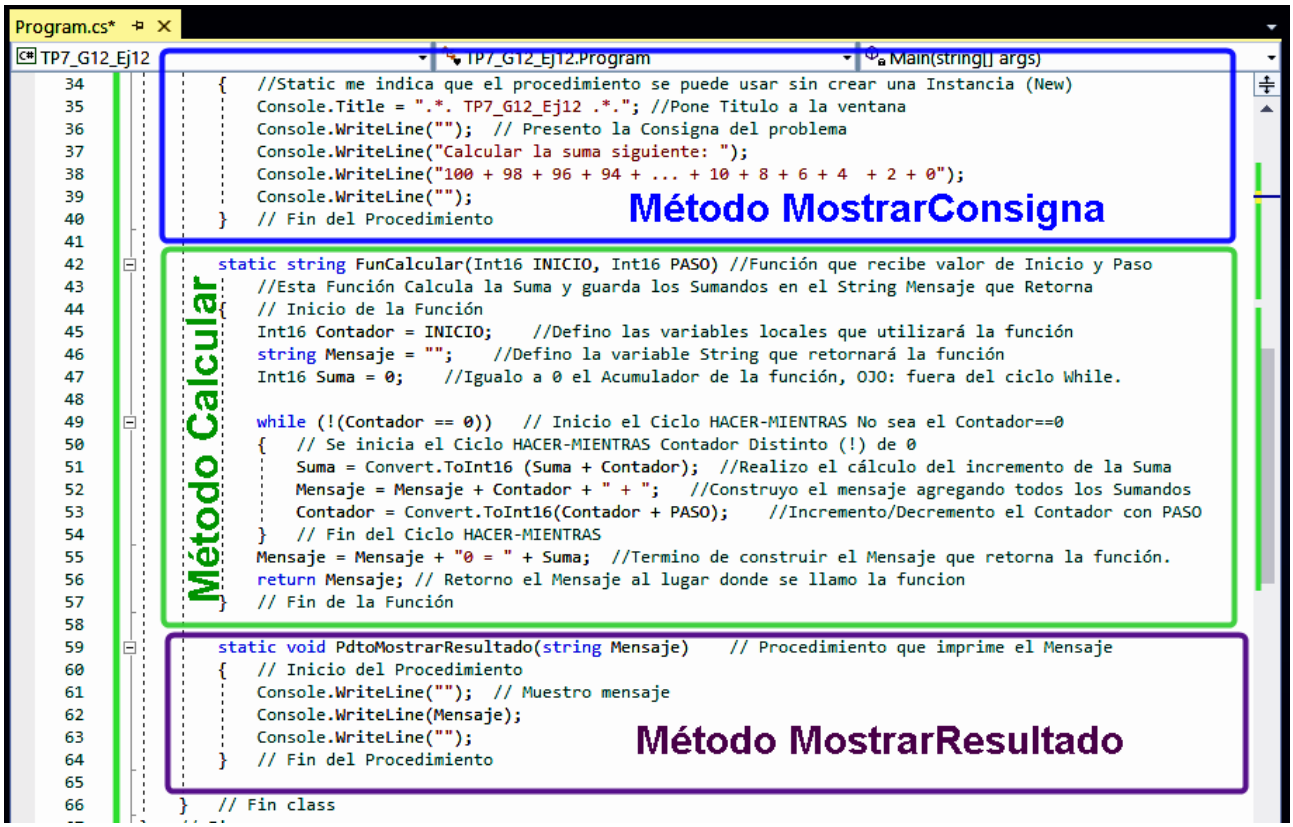
Como es fácil ver, la estructura del refinamiento progresivo empleada en Raptor a través de los Subprogramas, No se mantuvo al generar la conversión. Pero es conveniente, a medida que la complejidad de los programas avanza, mantener el diseño Top-Down también en VSC#; y eso lo hacemos a través de Métodos, que pueden ser: 1- Procedimientos (cuando llamamos a un subprograma que no nos retorna nada) y 2- Funciones (cuando nos retorna algún valor). En ambos casos podemos pasarle (o No) parámetros. Como ejemplo adjunto el código en VSC# del mismo programa, pero con los distintos trozos de código llevados a Métodos (los mismos que habíamos creado en Raptor)

Como puede verse, dentro del Método Main solo defino e inicializo las Constantes y Variables a utilizar. ***Luego llamo a tres métodos*** aún no creados y finalizo con las instrucciones para salir del programa.

Fuera del Main, realizo las tres definiciones de los métodos que uso en Main y copio dentro de ellos el código. Ver detalladamente el programa y prestar mucha atención a los comentarios que auto explican lo que se está realizando.




```
namespace TP7_G12_Ej12
{   // Inicio namespace. Espacio de Nombres igual al Nombre del Proyecto
    class Program
    {   // Inicio de la clase

        static void Main(string[] args) // Procedimiento (xq retorna void), y puede recibir muchos Argumentos tipo (string[])
        {   // Inicio función Main()

            // Defino e inicializo las CONSTANTES y variables a utilizar
            Int16 IntPASO = -2;
            Int16 IntINICIO = 100;
            string StrMensaje;

            PdtoMostrarConsigna();  //Llamo al procedimiento MostrarConsigna

            StrMensaje = FunCalcular(IntINICIO, IntPASO);   //Llamo a la Función Calcular

            PdtoMostrarResultado(StrMensaje);   //Llamo al procedimiento Mostrar Resultado.

            Console.WriteLine("Pulse cualquier tecla para Salir");  // Indico al Usuario una acción
            Console.ReadKey();  // Espero señal del Usuario para salir

        }   // Fin función Main()

//Fuera del Main defino los Procedimientos (No retornan NADA-void) y Funciones (Retornan algo)

        static void PdtoMostrarConsigna()  //Procedimiento que describe el Problema
        {   //Static me indica que el procedimiento se puede usar sin crear una Instancia (New)
            Console.Title = ".*. TP7_G12_Ej12 .*."; //Pone Titulo a la ventana
            Console.WriteLine("");  // Presento la Consigna del problema
```




```
34      {   //Static me indica que el procedimiento se puede usar sin crear una Instancia (New)
35          Console.Title = ".*. TP7_G12_Ej12 .*."; //Pone Titulo a la ventana
36          Console.WriteLine("");  // Presento la Consigna del problema
37          Console.WriteLine("Calcular la suma siguiente: ");
38          Console.WriteLine("100 + 98 + 96 + 94 + ... + 10 + 8 + 6 + 4  + 2 + 0");
39          Console.WriteLine("");
40      }   // Fin del Procedimiento
41
42      static string FunCalcular(Int16 INICIO, Int16 PASO) //Función que recibe valor de Inicio y Paso
43          //Esta Función Calcula la Suma y guarda los Sumandos en el String Mensaje que Retorna
44      {   // Inicio de la Función
45          Int16 Contador = INICIO;    //Defino las variables locales que utilizará la función
46          string Mensaje = "";    //Defino la variable String que retornará la función
47          Int16 Suma = 0;    //Igualo a 0 el Acumulador de la función, OJO: fuera del ciclo While.
48
49          while (!(Contador == 0))   // Inicio el Ciclo HACER-MIENTRAS No sea el Contador==0
50          {   // Se inicia el Ciclo HACER-MIENTRAS Contador Distinto (!) de 0
51              Suma = Convert.ToInt16 (Suma + Contador);  //Realizo el cálculo del incremento de la Suma
52              Mensaje = Mensaje + Contador + " + ";   //Construyo el mensaje agregando todos los Sumandos
53              Contador = Convert.ToInt16(Contador + PASO);    //Incremento/Decremento el Contador con PASO
54          }   // Fin del Ciclo HACER-MIENTRAS
55          Mensaje = Mensaje + "0 = " + Suma;  //Termino de construir el Mensaje que retorna la función.
56          return Mensaje; // Retorno el Mensaje al lugar donde se llamo la funcion
57      }   // Fin de la Función
58
59      static void PdtoMostrarResultado(string Mensaje)    // Procedimiento que imprime el Mensaje
60      {   // Inicio del Procedimiento
61          Console.WriteLine("");  // Muestro mensaje
62          Console.WriteLine(Mensaje);
63          Console.WriteLine("");
64      }   // Fin del Procedimiento
65
66  }   // Fin class
```

5) Codificación en C# (Windows From).

Procederemos ahora a realizar la solución del mismo problema, pero con Solución GUI (Interfaz Gráfica de Usuario) o sea, eligiendo Windows Forms.

Ingresamos a MicroSoft Visual Studio (MSVS), realizar Nuevo Proyecto y elegimos C# Windows Form (VSC# WF) Cuando se inicia el From1, lo primero que debemos hacer es cambiarle el ***Nombre*** al Form1 y la ***Propiedad Texto,*** para ponerle a la barra de Título de la ventana, el nombre que nosotros queramos. (En este proyecto ponemos TP7_G12_Ej12-WF)

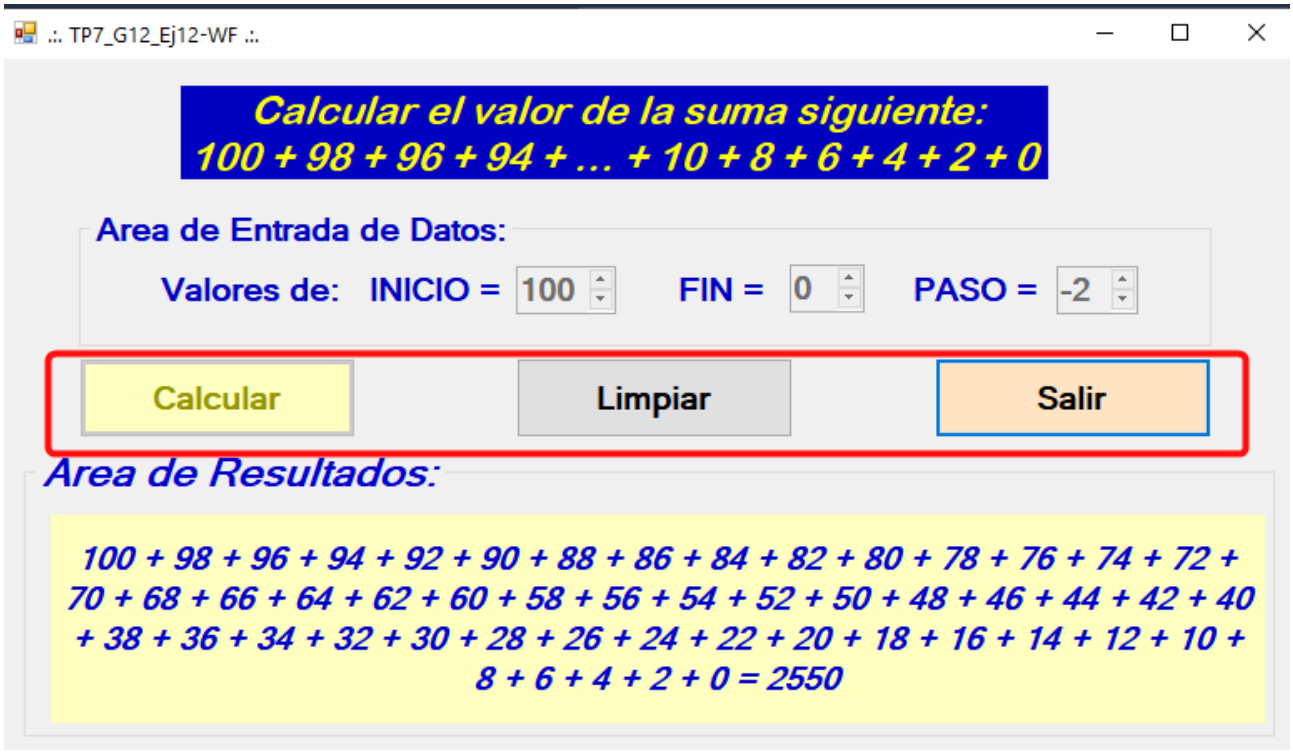
En la ventana Form1 diseñamos el Front-End

Usamos una ***Etiqueta*** para presentar la Consigna del programa. Realizamos cambios decorativos en las propiedades BackColor, ForeColor, así como **Font** Size, Bold, Italic, Name, etc.

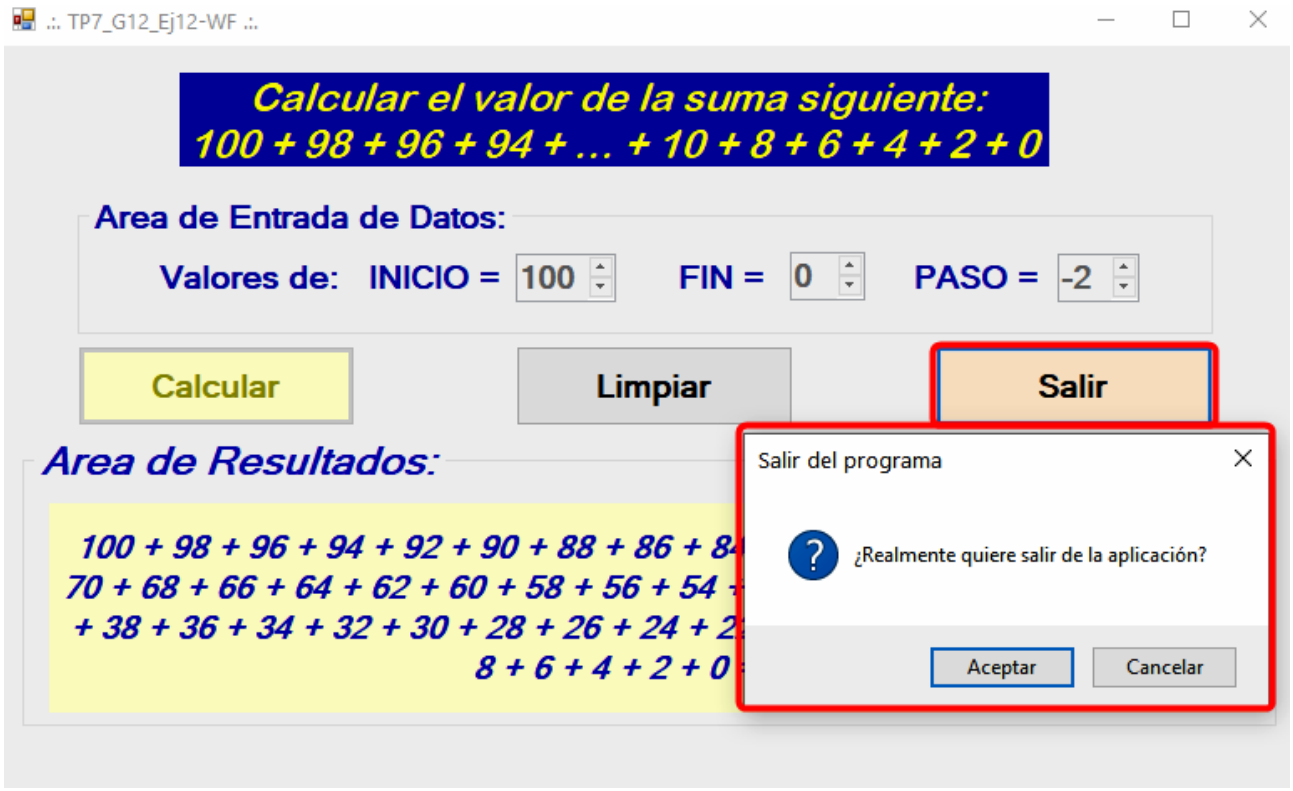
A continuación, dentro de un **groupBox** que denominamos Área de Entrada de Datos presentamos tres **Etiquetas** para indicar los valores a ingresar en c/u de los tres **NumericUpDown** (con la propiedad ***Enable = false,*** para que no puedan cambiarse) que usamos para cargar las CONSTANTES ya establecidas en la consigna; pero que si cambiamos a ***Enable = true,*** podemos liberar la carga de estos valores a gusto del usuario (previo control de que ***NudINICIO*** sea Distinto de ***NudFIN***, y de que si el ***NudPASO*** es positivo, el valor de INICIO sea menor que FIN, o a la inversa si el PASO es Negativo). Esta verificación la podemos realizar al iniciar el módulo **Calcular.**

Otra particularidad está dada por el comportamiento de los **Button**, que se inicia con el **BtnCalcular** seleccionado, el **BtnLimpiar** deshabilitado (simplemente porque NO hay nada que limpiar) y el **BtnSalir** habilitado. Esta situación cambia luego de hacer clic sobre el **BtnCalcular** ya que este

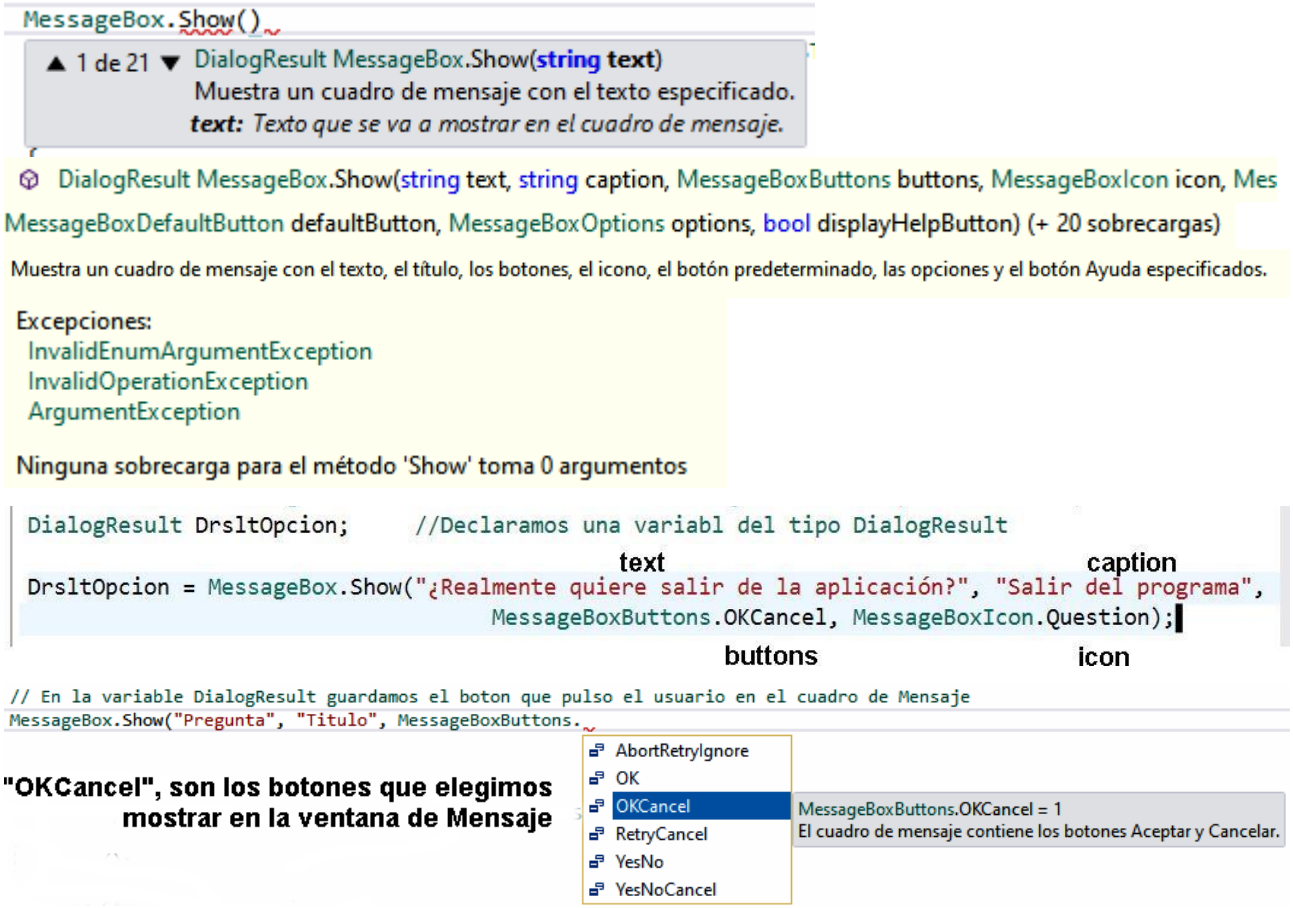
botón se deshabilita y aparece habilitado el botón **BtnLimpiar,** y si hacemos clic sobre este botón, la situación es como al inicio. Esto se logra con la propiedad ***Enable*** de los botones.



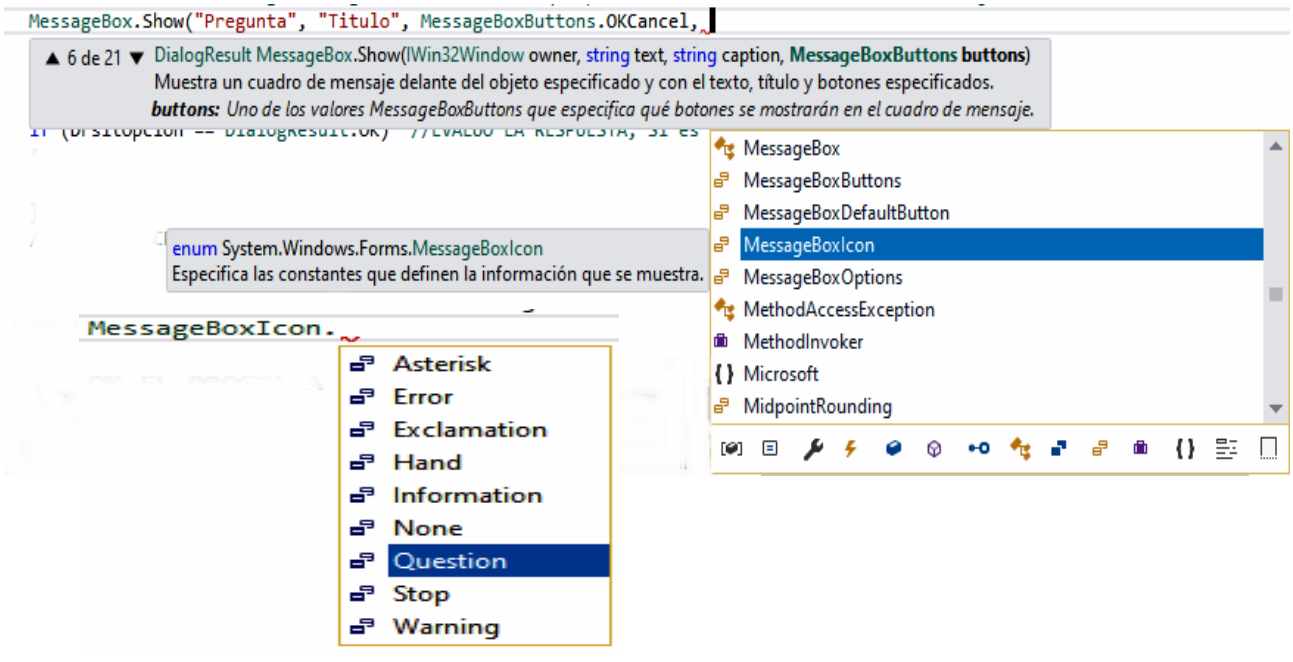
Otra característica *Nueva* para resaltar es que el programador considera que el usuario pueda pulsar equivocadamente el botón **Salir** y una vez pulsado se muestra una ventana de Mensaje **`MessageBox.Show`**, preguntándole al usuario si realmente quiere salir de la aplicación.



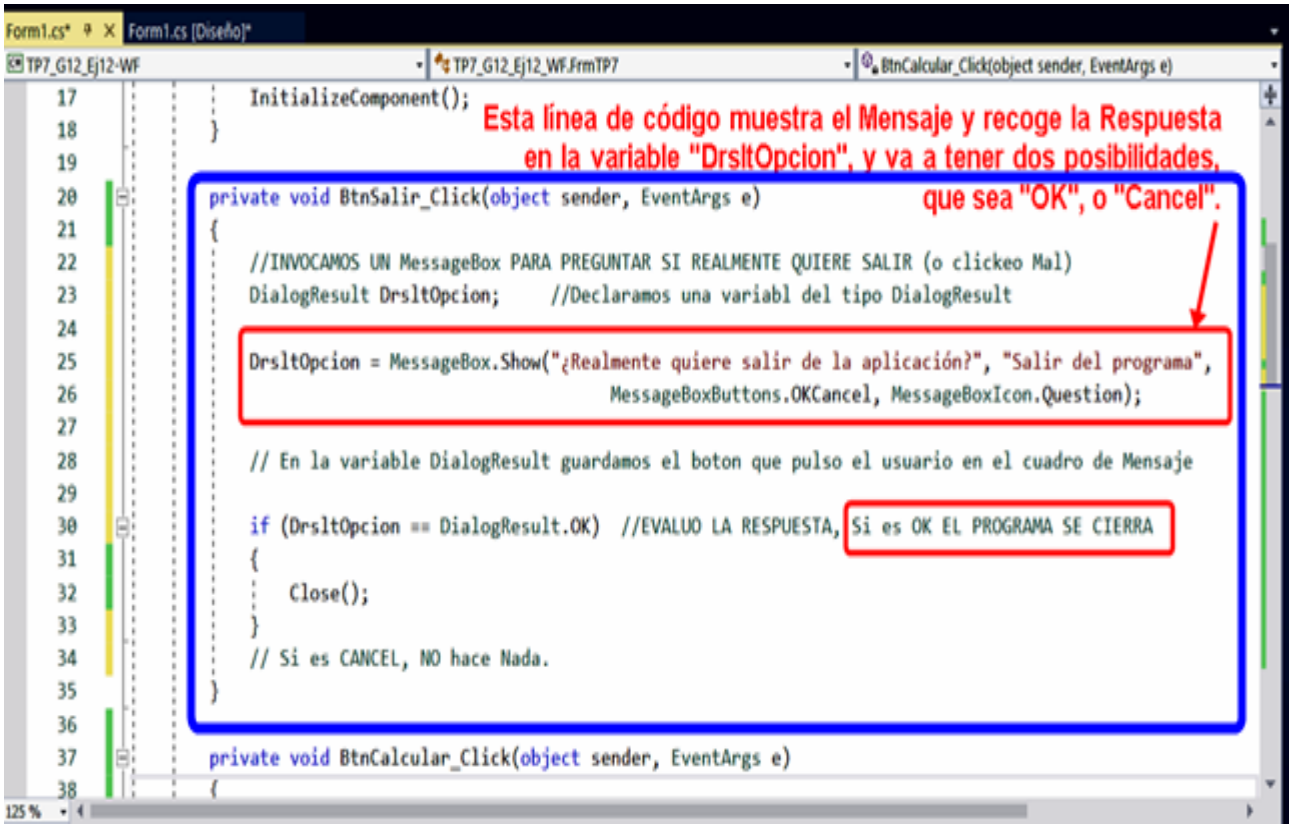
Esta ventana tiene dos botones, uno de **Aceptar (OK)** y otro de **Cancelar (Cancel)**



Luego de elegir los botones, ponemos una **“,”** y **“espacio”,** y nos aparecen estas indicaciones para elegir con que argumento seguimos recargando la función. Esta función tiene 21 sobrecargas (o distintas formas de armar la función según los parámetros que le carguemos. Nosotros no la aplicamos con todas sus opciones)

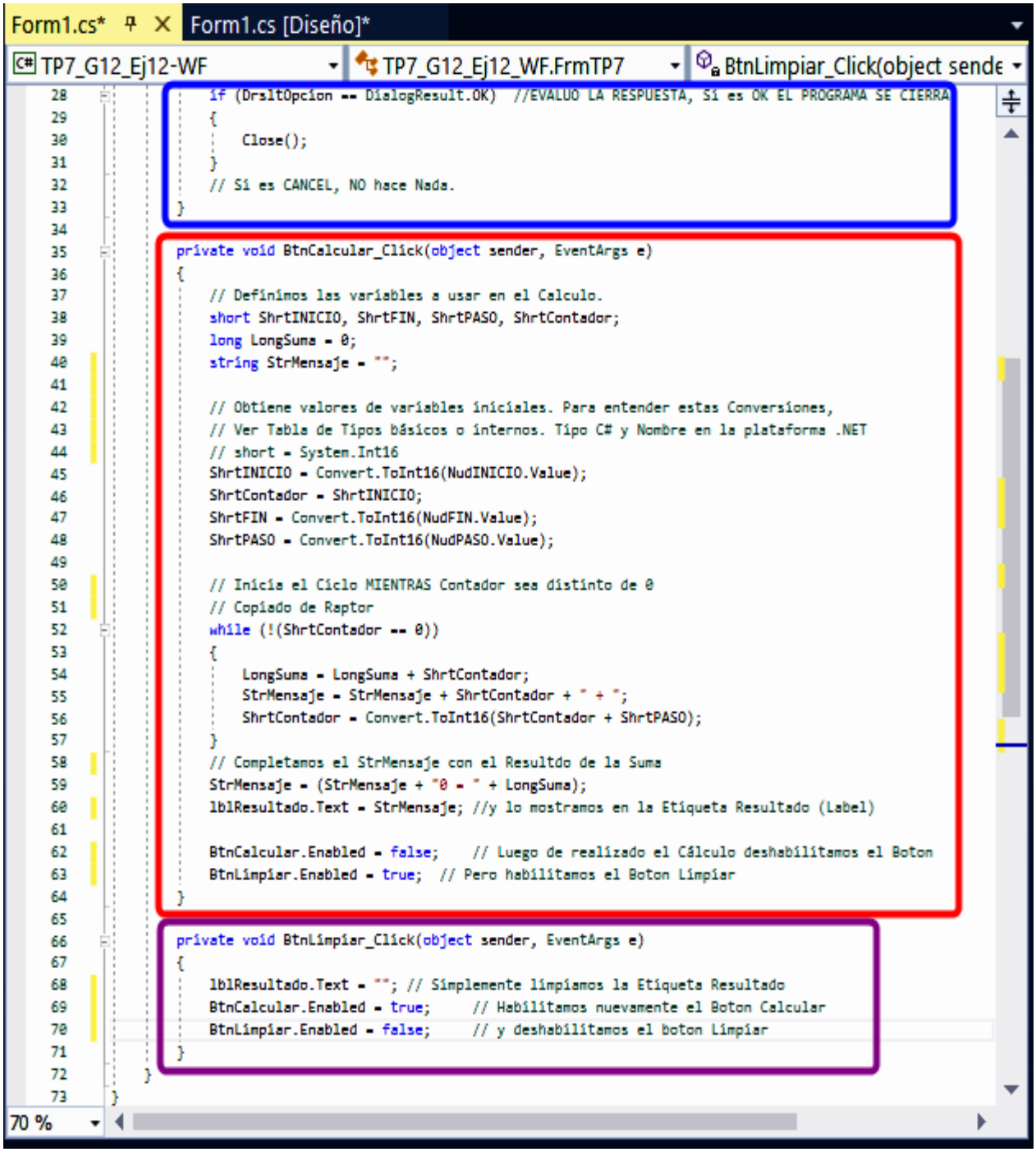


Finalmente, el Código completo del Método ***BtnSalir_Click***. (Aquí otra novedad que descubro en la versión 2017, prefiere que todos los Identificadores comiencen con Mayúsculas; y para que NO genere Mensajes o Advertencias, hemos adaptado la Notación Húngara a la nueva exigencia).



Por último, la ventana termina con otra **Etiqueta** que usamos para presentar el mensaje generado por el botón **Calcular.** Hemos cambiado varias propiedades del Control, para darles el aspecto que presenta.

Además, luego de mostrar el Resultado en la Etiqueta procedemos a cambiar las Habilitaciones de los botones Calcular y Limpiar




```
28      if (DrsltOpcion == DialogResult.OK)  //EVALUO LA RESPUESTA, Si es OK EL PROGRAMA SE CIERRA
29      {
30          Close();
31      }
32      // Si es CANCEL, NO hace Nada.
33  }
34
35  private void BtnCalcular_Click(object sender, EventArgs e)
36  {
37      // Definimos las variables a usar en el Calculo.
38      short ShrtINICIO, ShrtFIN, ShrtPASO, ShrtContador;
39      long LongSuma = 0;
40      string StrMensaje = "";
41
42      // Obtiene valores de variables iniciales. Para entender estas Conversiones,
43      // Ver Tabla de Tipos básicos o internos. Tipo C# y Nombre en la plataforma .NET
44      // short = System.Int16
45      ShrtINICIO = Convert.ToInt16(NudINICIO.Value);
46      ShrtContador = ShrtINICIO;
47      ShrtFIN = Convert.ToInt16(NudFIN.Value);
48      ShrtPASO = Convert.ToInt16(NudPASO.Value);
49
50      // Inicia el Ciclo MIENTRAS Contador sea distinto de 0
51      // Copiado de Raptor
52      while (!(ShrtContador == 0))
53      {
54          LongSuma = LongSuma + ShrtContador;
55          StrMensaje = StrMensaje + ShrtContador + " + ";
56          ShrtContador = Convert.ToInt16(ShrtContador + ShrtPASO);
57      }
58      // Completamos el StrMensaje con el Resultdo de la Suma
59      StrMensaje = (StrMensaje + "0 = " + LongSuma);
60      lblResultado.Text = StrMensaje; //y lo mostramos en la Etiqueta Resultado (Label)
61
62      BtnCalcular.Enabled = false;    // Luego de realizado el Cálculo deshabilitamos el Boton
63      BtnLimpiar.Enabled = true;  // Pero habilitamos el Boton Limpiar
64  }
65
66  private void BtnLimpiar_Click(object sender, EventArgs e)
67  {
68      lblResultado.Text = ""; // Simplemente limpiamos la Etiqueta Resultado
69      BtnCalcular.Enabled = true;     // Habilitamos nuevamente el Boton Calcular
70      BtnLimpiar.Enabled = false;     // y deshabilitamos el boton Limpiar
71  }
72  }
73  }
```

Para terminar, el código de los botones Limpiar y Calcular.

Enviar el Archivo ***TP7_Gx_Ejx.ZIP*** correspondiente al *Gx_Ejx. Resuelto* y Adjuntar también el Archivo: "***Auto_Evaluación TP-7 PeC 2020.xlsx***" completado para su Grupo