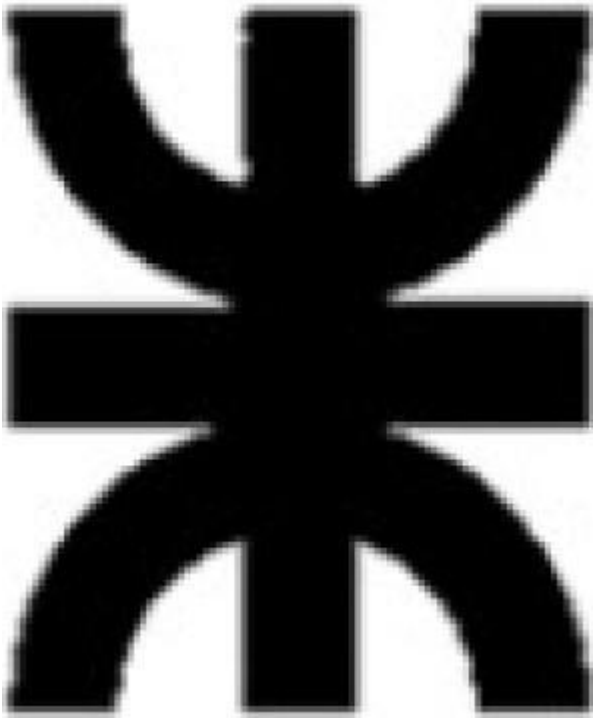


UNIVERSIDAD TECNOLÓGICA NACIONAL
Facultad Regional Reconquista

Programación en Computación

Ciclo Lectivo: 2020

Trabajo Práctico N.º 6

5 – Estructuras CONDICIONALES o de Selección

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

Guía Unidad 6: ALGORITMOS CONDICIONALES

SITUACIONES PROBLEMÁTICAS:

1. **Diseñe un algoritmo que convierta convertir una longitud dada en centímetros a pies.** Considere que 1 pie = 30.48 centímetros.
2. **Determinar si un alumno aprueba a reprueba un curso, sabiendo que aprobara si su promedio de tres calificaciones es mayor o igual a 70; reprueba en caso contrario.**
3. **Diseñe un algoritmo que determine el mayor de tres números. Considere que los tres números son diferentes entre sí.**
4. **Diseñar algoritmo para determinar el monto a pagar por un comprador, si en el caso de que el monto supere los $1.000, el comerciante le hace un 10 % de descuento por la compra total. Los datos ingresados son la cantidad y el precio unitario.**
5. **Redefinir el ejercicio anterior, teniendo en cuenta además que el comerciante decide hacerle el mismo descuento si compra más de 5 unidades del producto.**
6. **Para obtener un trabajo se requiere ser varón mayor de 18 años. Diseñe un algoritmo que decida si una persona puede calificar o no para el trabajo.**
7. **Una empresa ha decidido clasificar a sus empleados en cuatro grupos:**
   - Grupo 1: Solteros con menos de 25 años.
   - Grupo 2: Solteros con 25 años a más.
   - Grupo 3: Casados con menos de 34 años.
   - Grupo 4: Casados con 34 años a más.

   Dar un mensaje informando el grupo al que pertenece el empleado.
8. **Un obrero necesita calcular su salario semanal, el cual se obtiene de la siguiente manera:**
   - Si trabaja 40 horas o menos se le paga $16 por hora
   - Si trabaja más de 40 horas se le paga $16 por cada una de las primeras 40 horas y $20 por cada hora extra.
9. **Determinar la cantidad de dinero que recibirá un trabajador por concepto de las horas extras trabajadas en una empresa, sabiendo que cuando las horas de trabajo exceden de 40, el resto se consideran horas extras y que estas se pagan al doble de una hora normal cuando no exceden de 8; si las horas extras exceden de 8 se pagan las primeras 8 al doble de lo que se pagan las horas normales y el resto al triple.**
10. **Diseñe un algoritmo que lea tres longitudes y determine si forman o no un triángulo. Si es un triángulo determine de qué tipo de triángulo se trata entre: equilátero (si tiene tres lados iguales), isósceles (si tiene dos lados iguales) o escaleno (si tiene tres lados desiguales).** Considere que para formar un triángulo se requiere que: ***"el lado mayor sea menor que la suma de los otros dos".***
11. **Diseñe un algoritmo que determine quién tiene la mayor Edad entre Juan, Mario y Pedro. Considere que dos pueden tener la mayor edad, o ser contemporáneos los tres.**
12. **En una playa de estacionamiento de vehículos se cobra $ 50 por hora o fracción. La medición del tiempo se efectúa anotando la hora de entrada y la hora de la salida, ambas en el formato HH:MM, según un reloj de 24 horas. Diseñe un algoritmo para determinar el monto que debe pagar un cliente por el estacionamiento de su vehículo. Considere que tanto la hora de entrada como de salida corresponden al mismo día.**

ACTIVIDADES:

Resolver las situaciones problemáticas anteriores, mediante pseudocódigo, Diagrama de Flujo en RAPTOR y codificación en C# (Consola y/o Ventanas).

ANALIZAR:

- *.Datos de Entrada, Salida y Auxiliares (si son necesarios)*
- *¿Que ocurre sí? Y determinar el proceso necesario*

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. Resolución de problemas usando estructuras secuenciales para la formulación de sus algoritmos. U_6) Estructuras de selección o condicionales, estructuras de selección múltiple. Uso de condicionales para la formulación de 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:

En una playa de estacionamiento de vehículos se cobra $ 50 por hora o fracción. La medición del tiempo se efectúa anotando la hora de entrada y la hora de la salida, ambas en el formato HH:MM, según un reloj de 24 horas. Diseñe un algoritmo para determinar el monto que debe pagar un cliente por el estacionamiento de su vehículo. Considere que tanto la hora de entrada como de salida corresponden al mismo día.

1) Pseudo-Código:

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

algoritmo <*Costo_Estacionamiento*>

// determina el monto que debe pagar un cliente

// por el estacionamiento de su vehículo

// Fecha de Entrega: DD/MM/AA.

// Programador: Grupo 12 de PeC.

Variables **// definiciones de las variables y de su tipo.**

Entero eHoraEntrada, eMinutoEntrada, eHoraSalida, eMinutoSalida;
// entero Hora y Minutos de Entrada y Hora y Minutos de Salida

Entero eTiempoMinutos, eHorasAPagar, eResiduoEnMinutos;

// Tiempo de Estacionamiento en Minutos, Horas a Pagar y

// Residuo en Minutos de eTiempoMinutos **entre 60'**

Real rMontoAPagar; **// real Monto a Pagar.**

INICIO **// Inicio del Pseudocódigo**

Leer (eHoraEntrada, eMinutoEntrada, eHoraSalida, eMinutoSalida);

// Ingresa los Datos

// Realiza el proceso de Cálculo del tiempo en minutos

Asignar eTiempoMinutos ← (eHoraSalida – eHoraEntrada) * 60 + (eMinutoSalida – eMinutoEntrada);

// Determina la cantidad entera de horas a pagar
// Si sobran minutos, se cobran como una hora adicional.

Asignar eHorasAPagar ← (eTiempoMinutos / 60);

// Determina la cantidad entera de horas.

Asignar eResiduoEnMinutos ← (eTiempoMinutos % 60);

// Determina la cantidad de minutos que sobran, o fracción de hora

Si (eResiduoEnMinutos ¡= 0) **// Si sobran minutos**

eHorasAPagar ← (eHorasAPagar + 1)

FinSi

// Determina el monto a Pagar:

Asignar rMontoAPagar ← eHorasAPagar * 50;

Imprimir: **(*"El Importe a pagar es de $: "*** + rMontoAPagar**)**;

FIN **// Fin del Pseudocódigo**

Los espacios en blanco no son significativos, pero se los suele usar para separar las partes importantes. Los elementos léxicos del seudocódigo son:

a) Comentarios: documentan el algoritmo con anotaciones sobre su funcionamiento. Es mejor que sobren a que falten, ya que ayudan a seguir el Programa.

b) Palabras Reservadas: son palabras con significado especial como **inicio** y **fin.**

c) Identificadores: son los nombres que se dan a los objetos (variables, funciones, etc.) que manipula el algoritmo, y deben seguir ciertas reglas:

- El nombre debe resultar significativo.
- No podrá coincidir con Palabras Reservadas.
- Comenzará siempre con un carácter alfabético.

d) Operadores: Los operadores se utilizan en las Expresiones e indican las operaciones a efectuar con los operandos.

e) Signos de puntuación: Los signos de puntuación se emplean con el objeto de agrupar o separar. Ej.: **. , ; ( ) [ ] etc.**

f) Literales: son valores que aparecen escritos directamente en el programa (entero, real, lógicos, string, etc.)

Las tres estructuras básicas tienen su propia representación:

- **Secuenciales.** Ejecuta todas las instrucciones en el orden dado.
- **Selectivas o Condicionales.** Se ejecutan en función de una condición
- **Repetitivas.** Repiten una porción de código en base a una condición.

2) Diagramas de Flujo.

Utiliza **símbolos normalizados** unidos por flechas

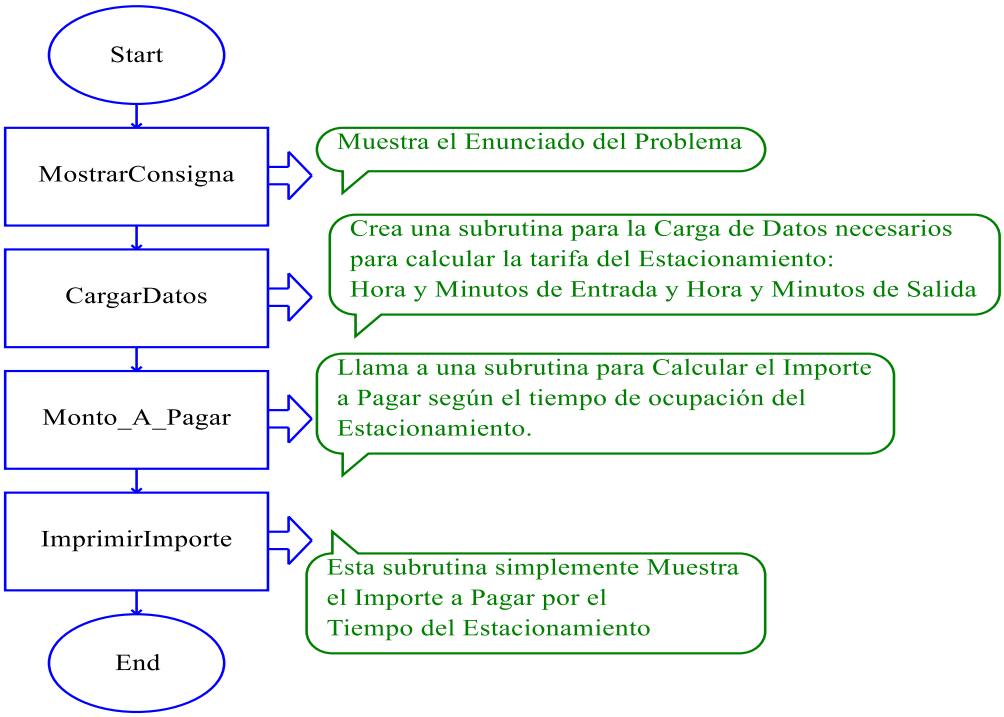
(**líneas de flujo**) que indican el orden en que debe

ejecutarse cada paso.

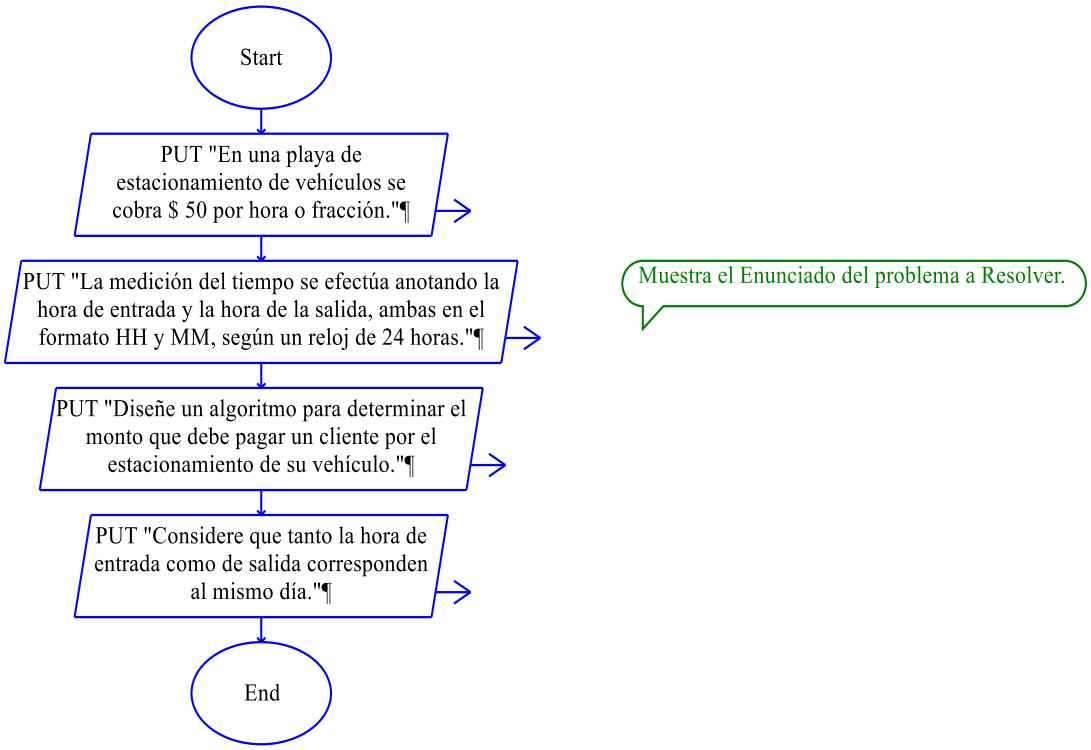
Es muy importante porque permite:

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

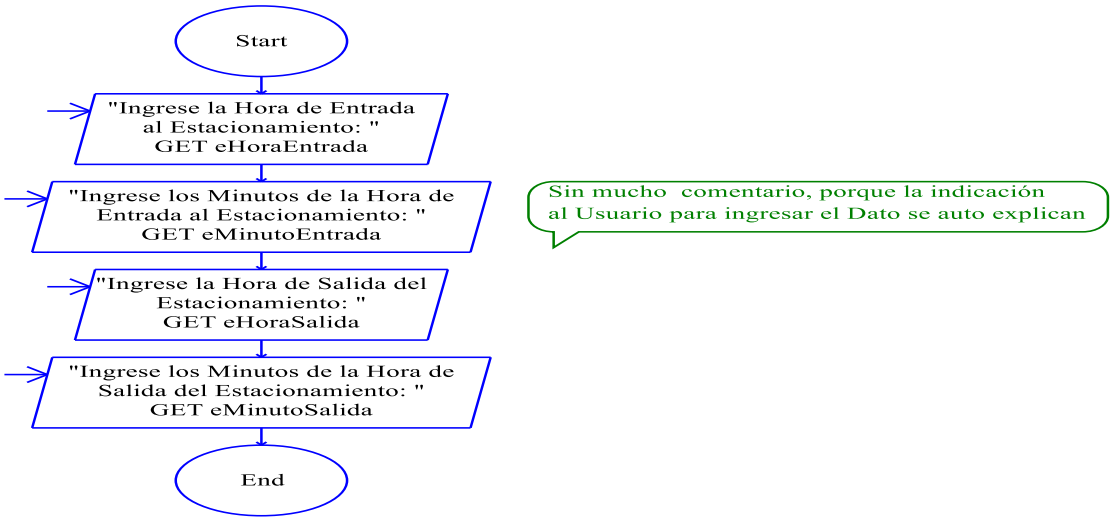
Programa **“main”**



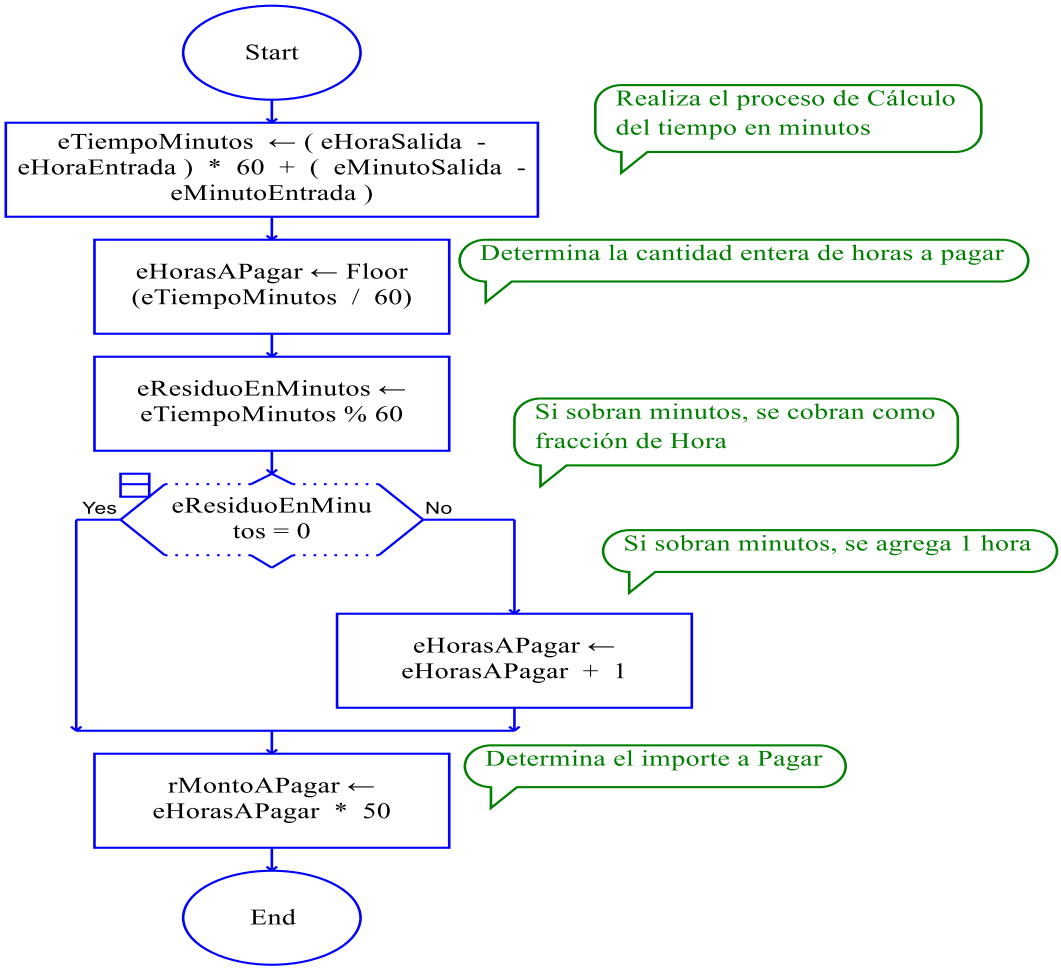
Subrutina **“MostrarConsigna”**



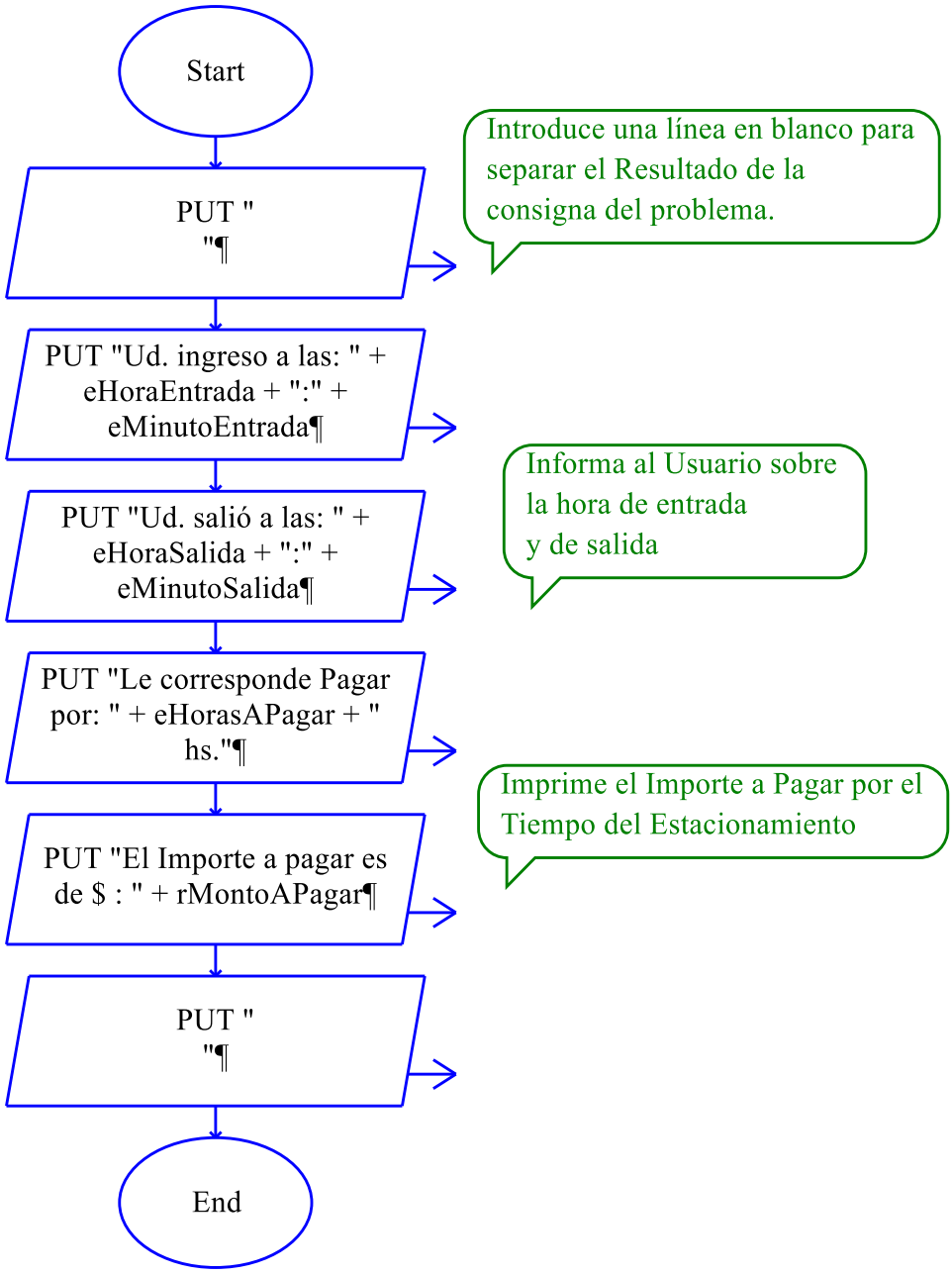
Subrutina **“CargarDatos”**



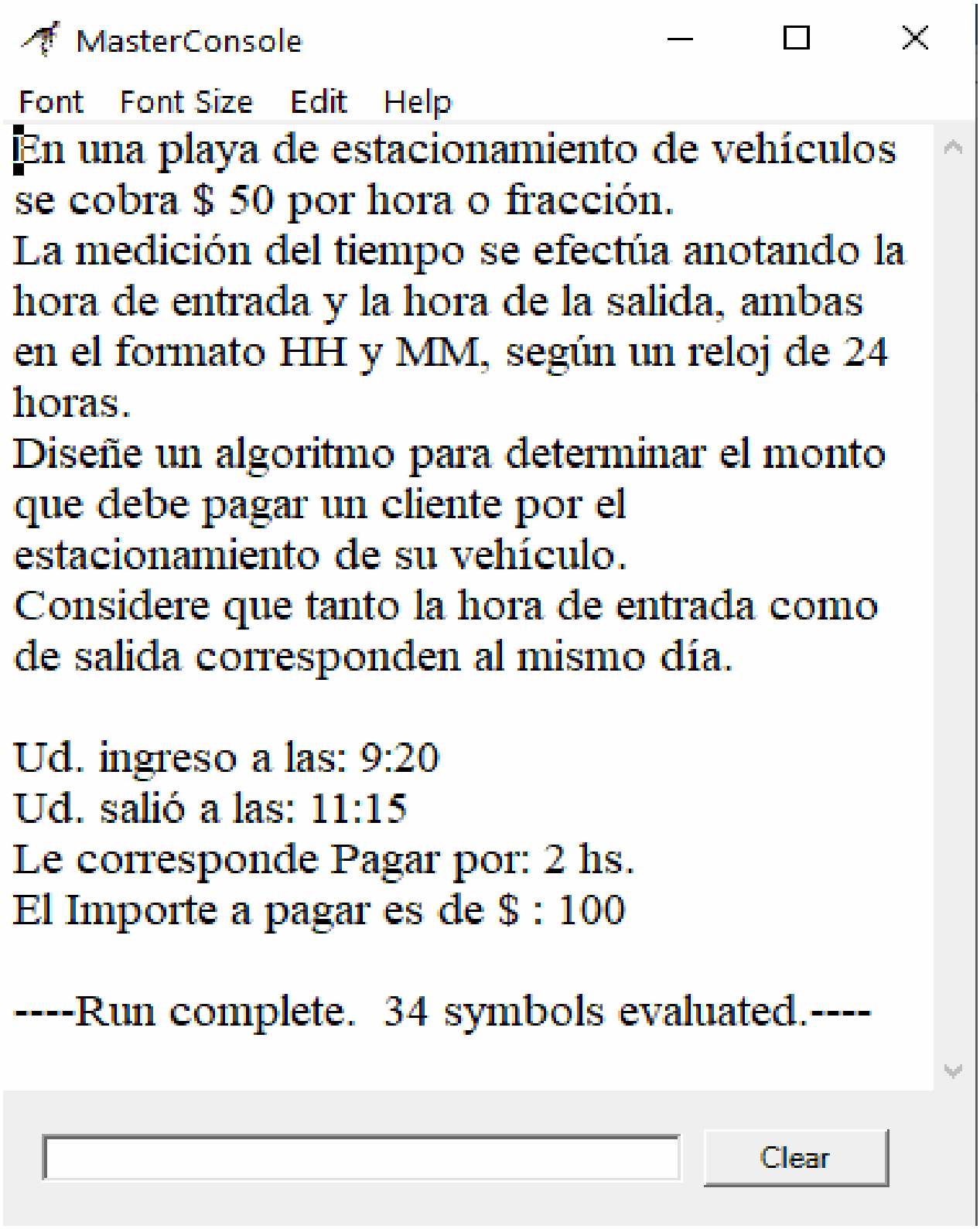
Subrutina **“Monto_A_Pagar”**



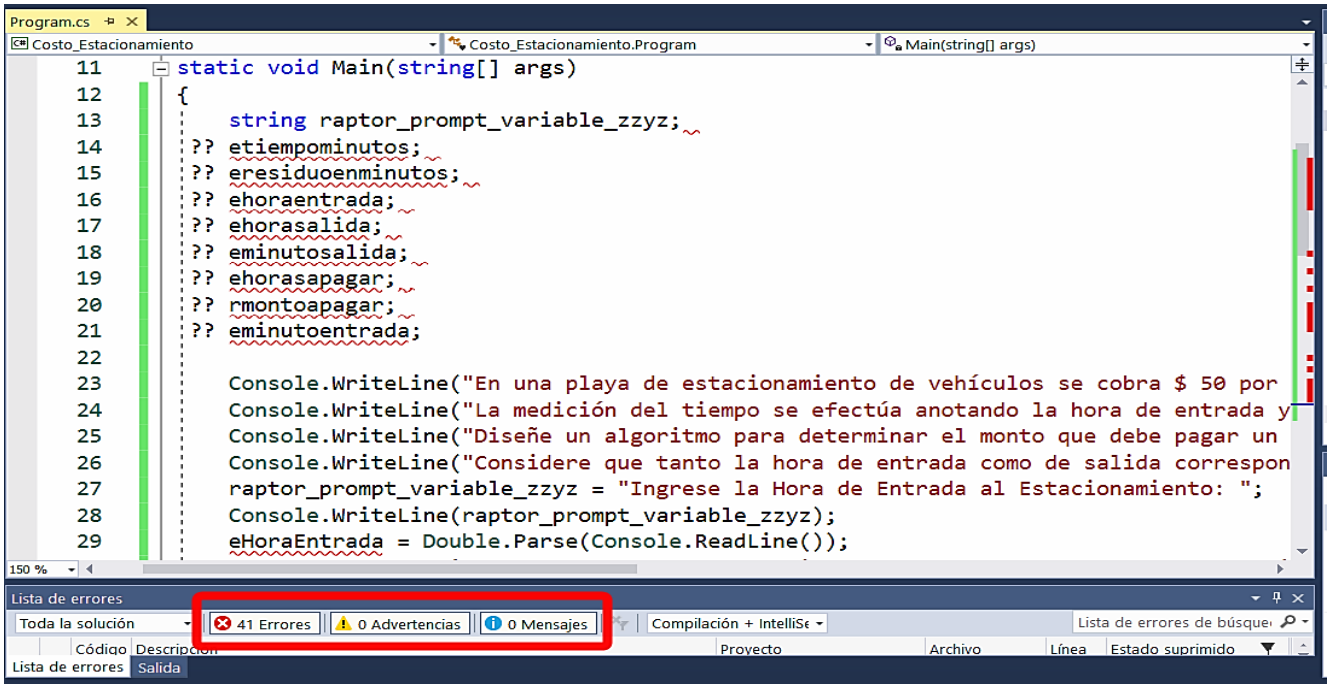
Subrutina **“ImprimirImporte”**



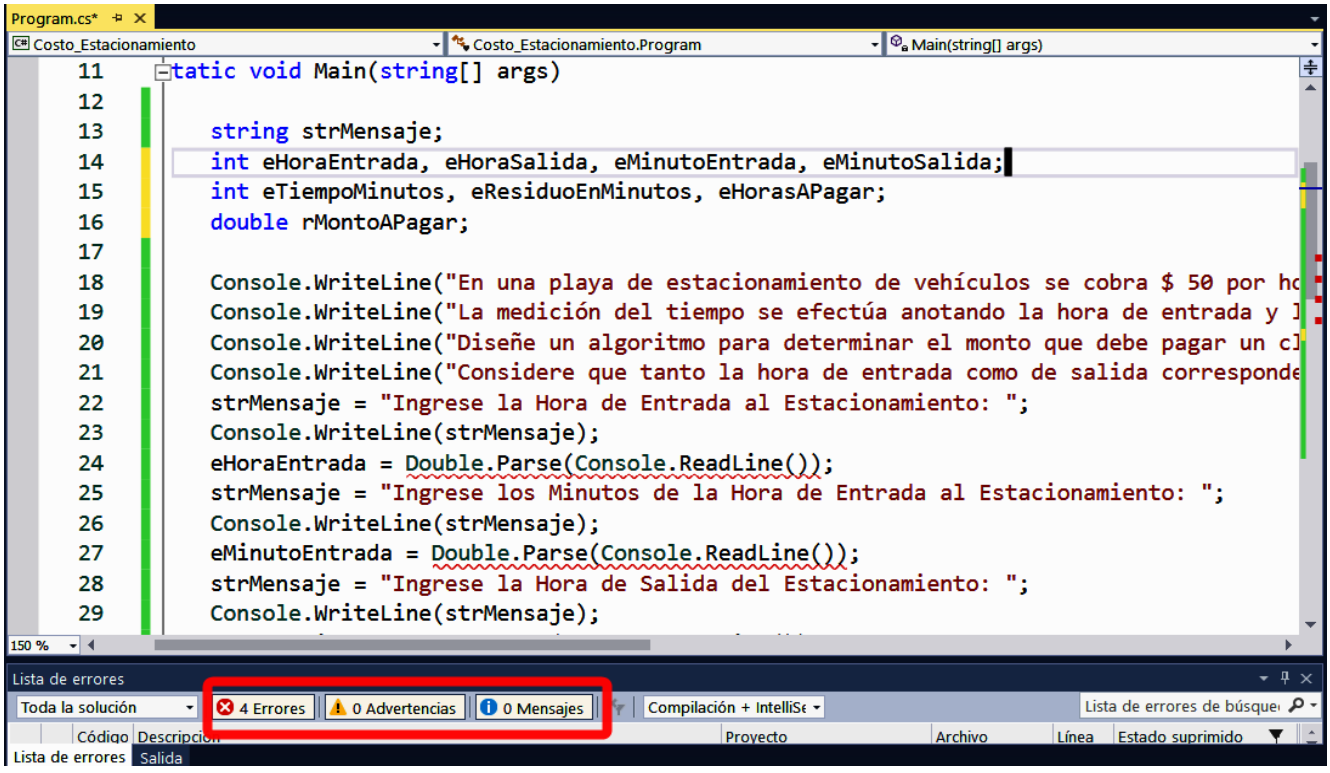
Como se puede ver, hemos realizado el Diagrama de Flujo (D.F.) en Raptor, y hemos realizado el programa dentro de los símbolos del D.F. Hemos ejecutado la aplicación y podemos ver los resultados que nos muestra en la Consola de Salida de Raptor.


MasterConsole
Font Font Size Edit Help
En una playa de estacionamiento de vehículos se cobra $ 50 por hora o fracción.
La medición del tiempo se efectúa anotando la hora de entrada y la hora de la salida, ambas en el formato HH y MM, según un reloj de 24 horas.
Diseñe un algoritmo para determinar el monto que debe pagar un cliente por el estacionamiento de su vehículo.
Considere que tanto la hora de entrada como de salida corresponden al mismo día.
Ud. ingreso a las: 9:20
Ud. salió a las: 11:15
Le corresponde Pagar por: 2 hs.
El Importe a pagar es de $: 100
----Run complete. 34 symbols evaluated.----
Clear

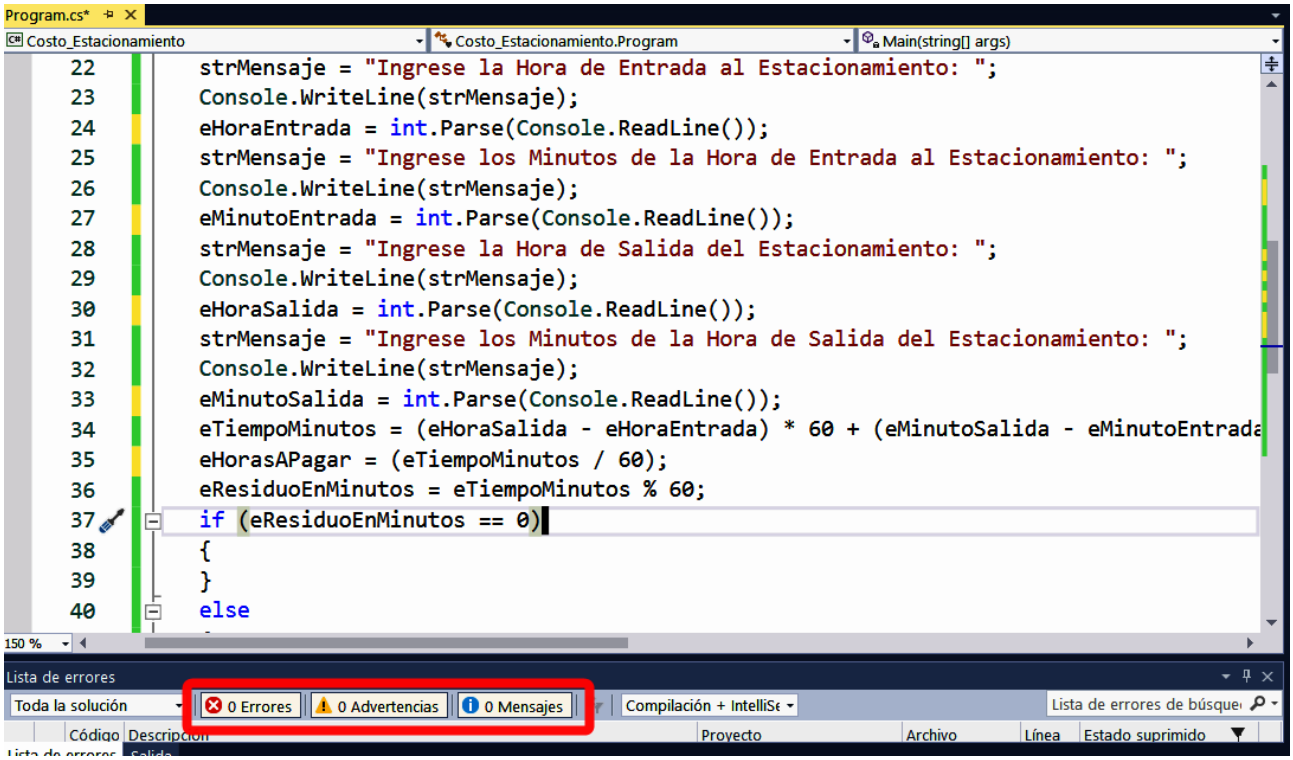
Ver la copia del código C# generado por Raptor y cargado en Visual C# en la imagen siguiente:



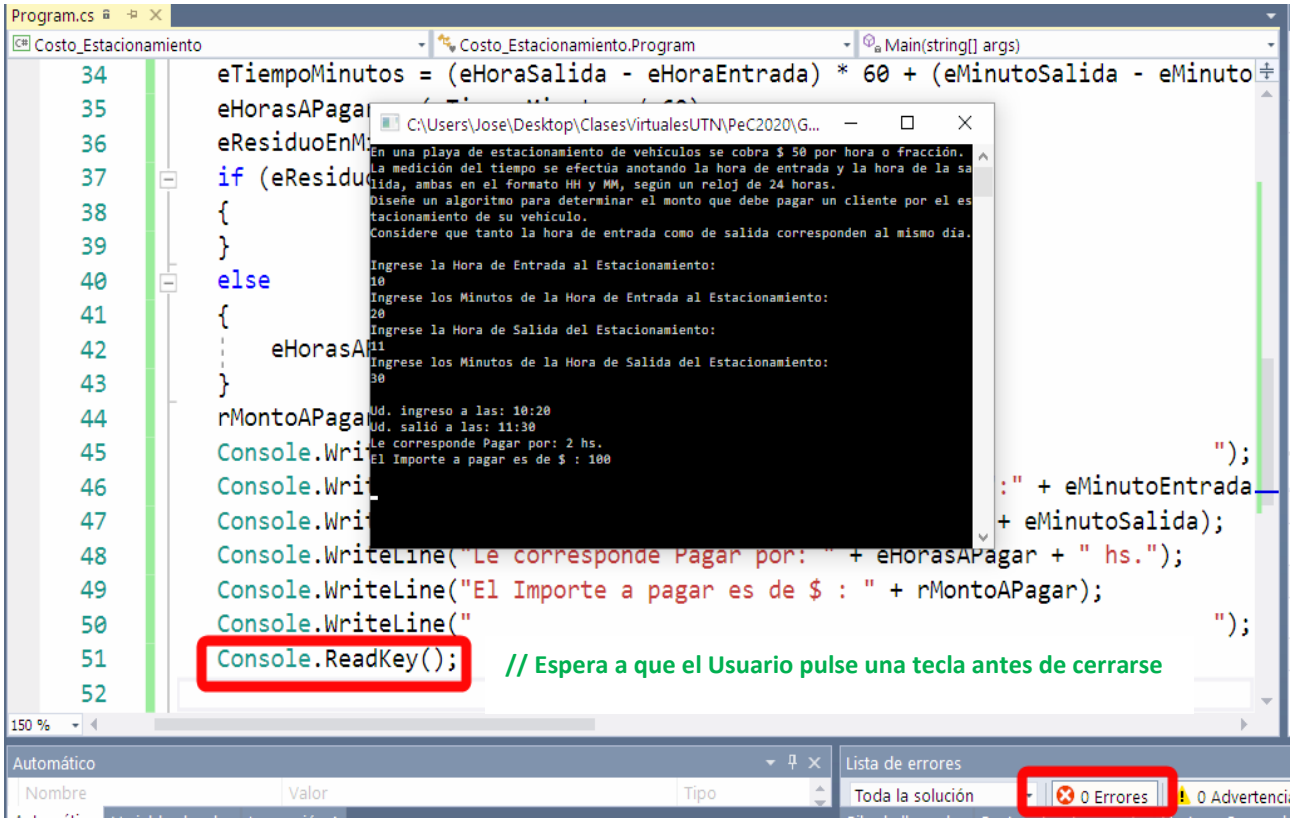
Como era de esperar, la cantidad de Errores es alta (41) y no tenemos Advertencias ni Mensajes, pero corrigiendo solo la definición de las variables, ya estamos en solo 4 Errores



Y cambiando la función de convertir las entradas del usuario por teclado de **Double** a **int,** ya no tenemos más errores y podemos compilar el programa.



Programa ejecutándose:



El programa se está ejecutando en el modo consola del S.O. Windows (Ventana Negra con letras blancas). El código final del programa (con comentarios agregados – las líneas verdes) lo pueden completar Uds. Copiando los comentarios ya realizados en Raptor.

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#.***

Conceptos de Repaso: Tipos de Datos más usados en C#.

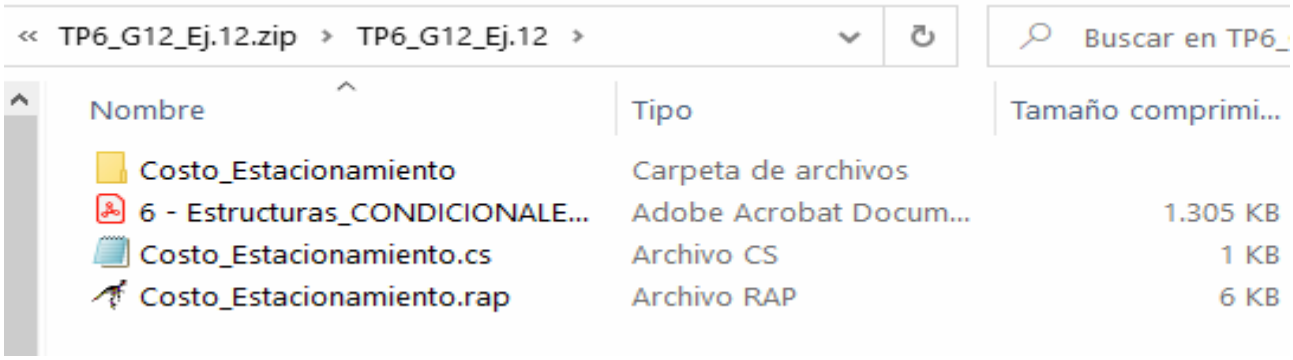
Tabla de *Tipos básicos o internos* [S = Signed (o con signo), U = Unsigned (o sin signo)**].**

Tipo C#	Nombre en la plataforma .NET	¿Con signo?	Bytes utilizados	Valores que soporta
bool	System.Boolean	No	1	true o false (verdadero o falso en inglés)
byte	System.Byte	No	1	0 hasta 255
sbyte	System.SByte	Si	1	-128 hasta 127
short	System.Int16	Si	2	-32.768 hasta 32.767
ushort	System.Uint16	No	2	0 hasta 65535
int	System.Int32	Si	4	-2.147.483.648 hasta 2.147.483.647
uint	System.Uint32	No	4	0 hasta 4.394.967.395
long	System.Int64	Si	8	-9.223.372.036.854.775.808 hasta 9.223.372.036.854.775.807
ulong	System.Uint64	No	8	0 hasta 18446744073709551615
float	System.Single	Si	4	Aproximadamente ±1.5E-45 hasta ±3.4E38 con 7 cifras significativas
double	System.Double	Si	8	Aproximadamente ±5.0E-324 hasta ±1.7E308 con 7 cifras significativas
decimal	System.Decimal	Si	12	Aproximadamente ±1.0E-28 hasta ±7.9E28 con 28 ó 29 cifras significativas
char	System.Char		2	Cualquier carácter Unicode (16 bits)

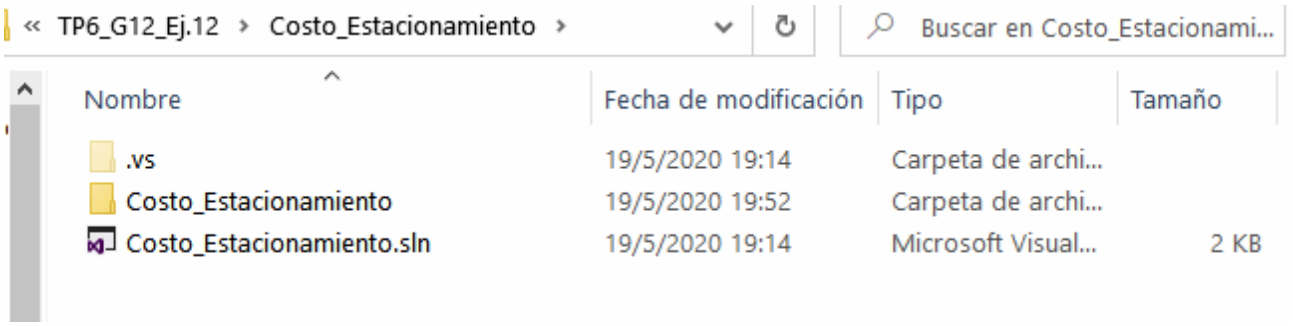
Escogiendo qué tipo usar

A la hora de programar deberéis decidir qué tipo de variables querréis usar. Generalmente esta decisión se basa en el tipo de información que vayáis a usar y en el tamaño de la información. Por ejemplo, si necesitamos hacer la suma de dos valores numéricos enteros, (usando la palabra clave **int**) los cuales de acuerdo con nuestra tabla son números enteros (no pueden llevar valores decimales) y podrán aceptar valores entre -2,147,483,648 y 2,147,483,647 lo cual es más que suficiente para nuestro ejemplo de sumar dos números. En el caso de que necesitáramos hacer uso de números reales (los cuales poseen una parte entera y una parte decimal como el número 10.22) podremos hacer uso del tipo **float**, **double** y **decimal** de acuerdo con el tamaño del número que necesitemos y así cada uno de los tipos tiene su uso y capacidad de acuerdo con la tabla.

El archivo a enviar, “TP6_G12_Ej.12.zip” contiene hasta este momento:

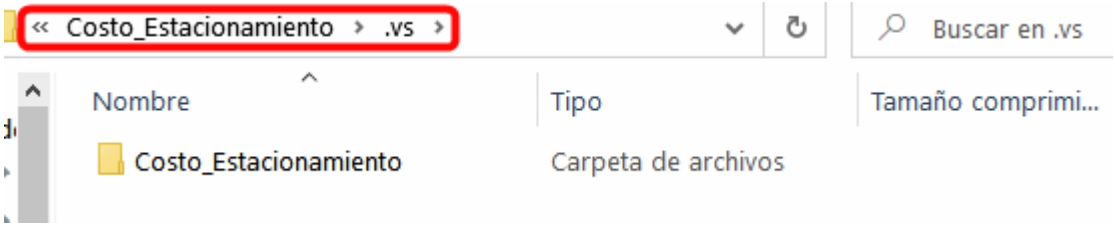


Y la carpeta “Costo_Estacionamiento”, contiene otra carpeta con el mismo nombre, además del archivo “Costo_Estacionamiento.sln”, que es el que abre el Proyecto; una carpeta “.vs” (el “.” Indica que es una carpeta oculta, o sea, depende de como está configurado el Windows Explorer para que se muestre o no.

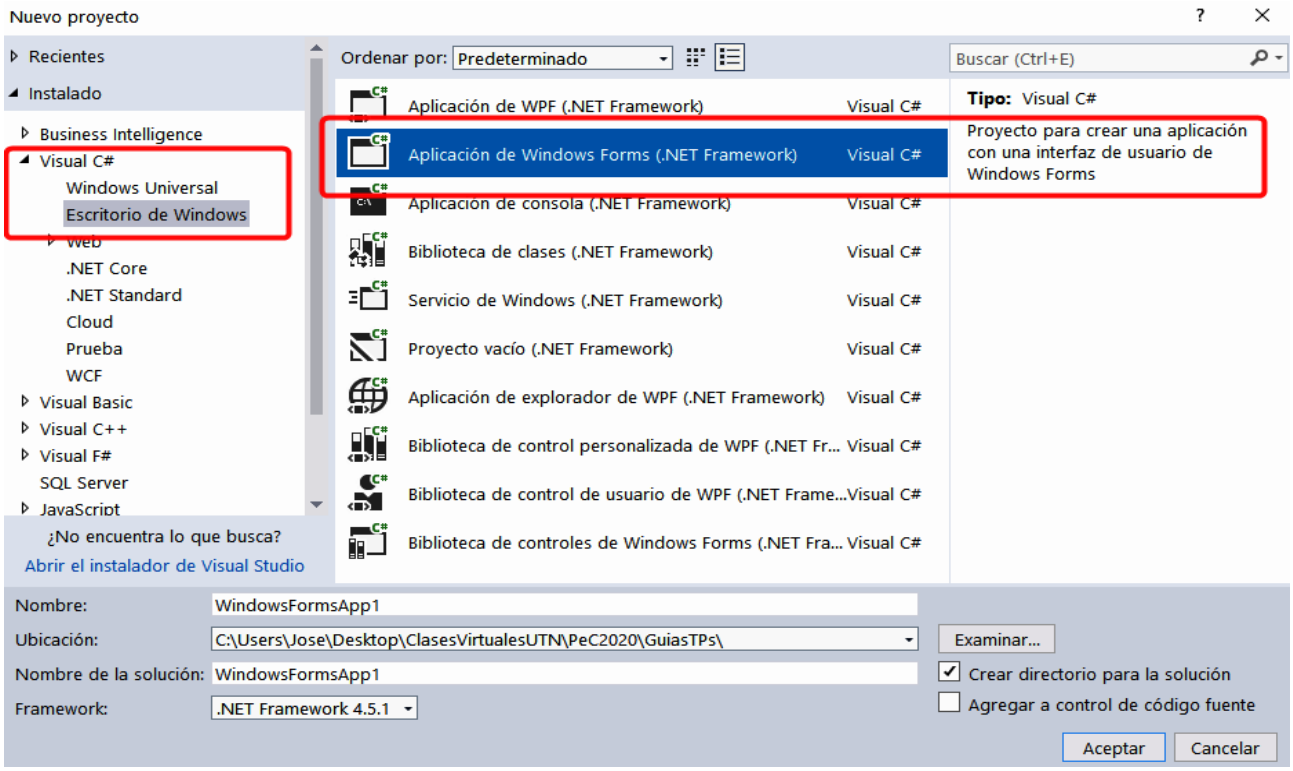


La segunda carpeta “Costo_Estacionamiento”, contiene el proyecto generado por Visual C#

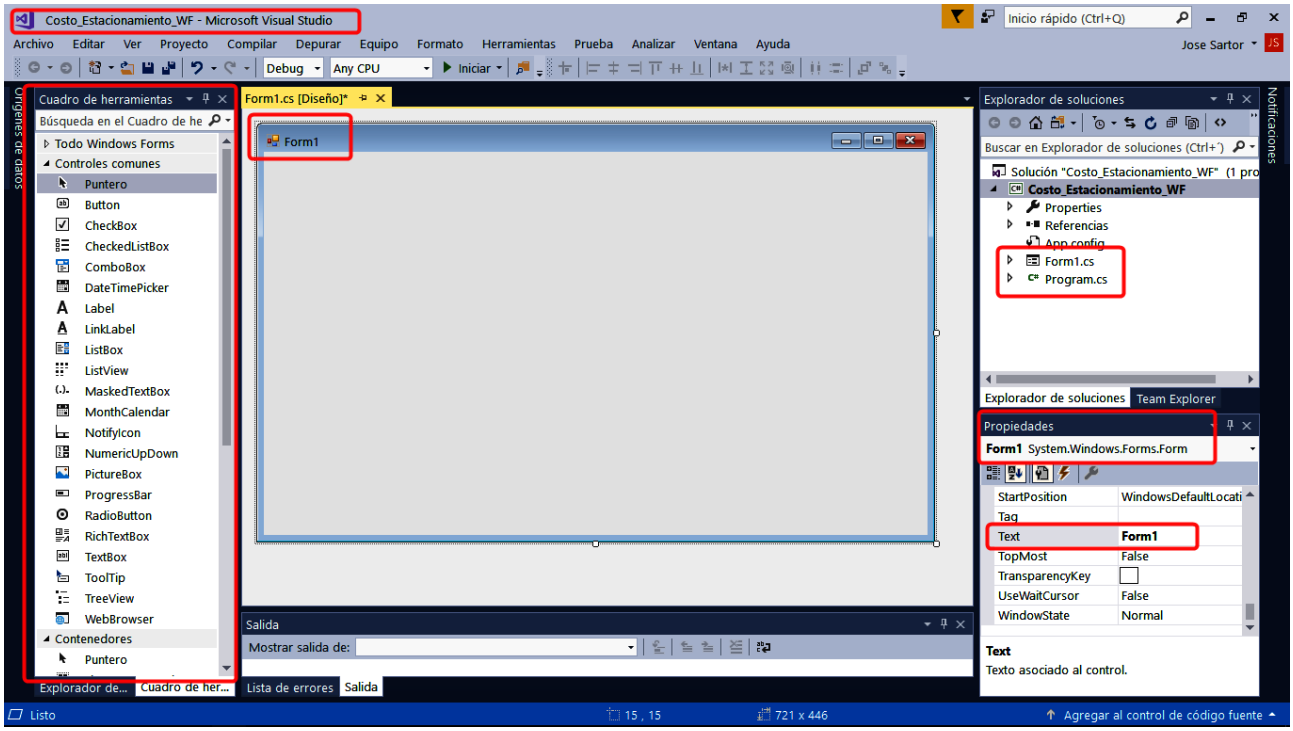
La Carpeta oculta “.vs”, contiene dentro otra carpeta “Costo_Estacionamiento”, con otras carpetas con directivas para el compilador.



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



En el proyecto de Windows Forms (WF), veremos una ventana central que incluye el Formulario (Form1):



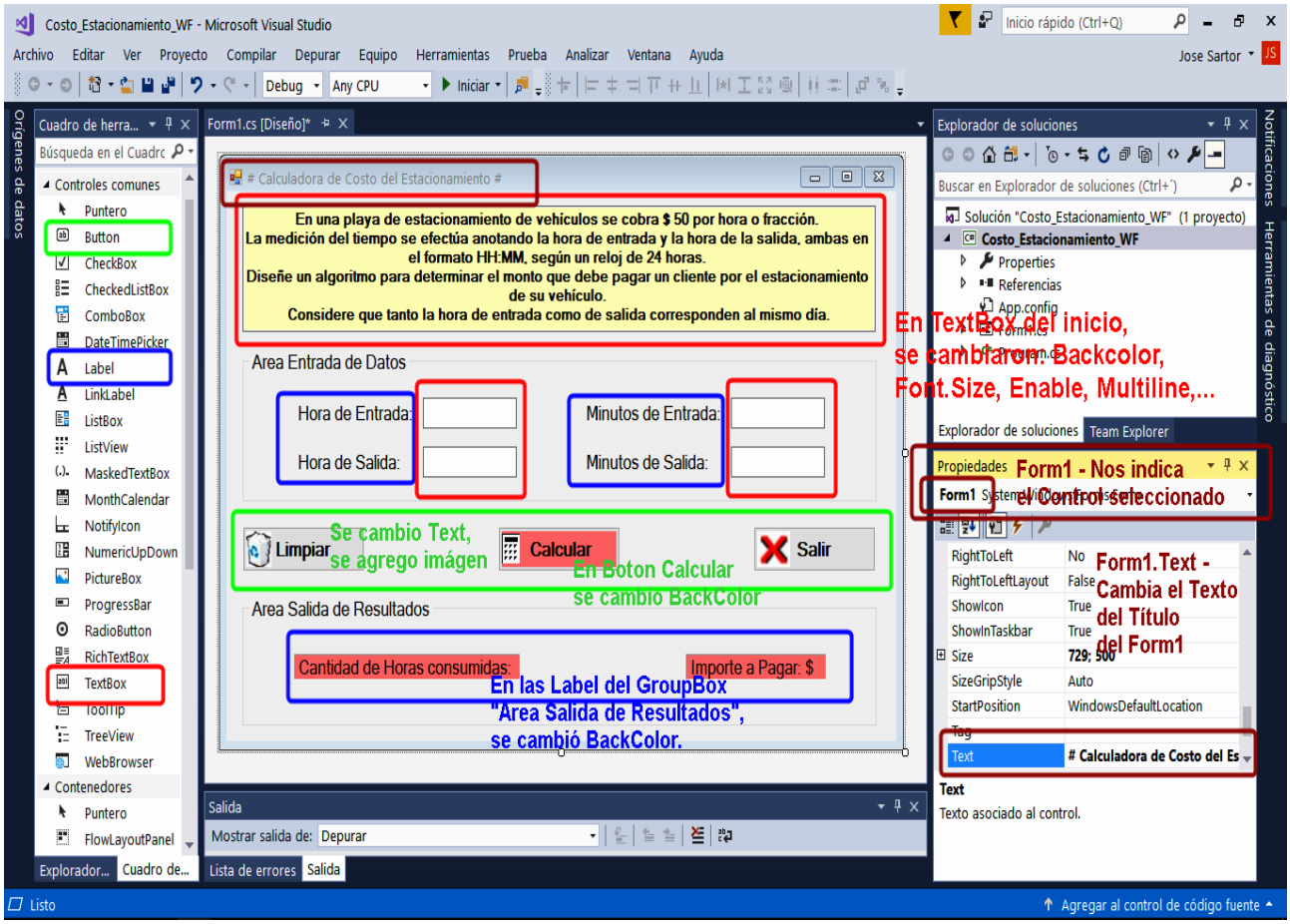
El formulario es la ventana de nuestra aplicación, y antes de ir a escribir código, debemos Diseñar la ventana del programa. Para dibujar los elementos o ***Controles*** en el Formulario, a la **Izquierda** tenemos la Ventana del ***Cuadro de Herramientas,*** que contiene todos los ***controles*** que podemos dibujar en el Formulario.

Es aconsejable dividir la ventana en tres partes o áreas, y dentro de un Contenedor c/u de ellas:

1. ***Área de Carga de Datos*** (Generalmente formado por Etiquetas y Cajas de Texto)
2. ***Área de Procesos*** (Botones de comando, que son los que contienen el código)
3. ***Área de Salida*** (Destinada a mostrar los resultados en Etiquetas o Cajas de Texto)

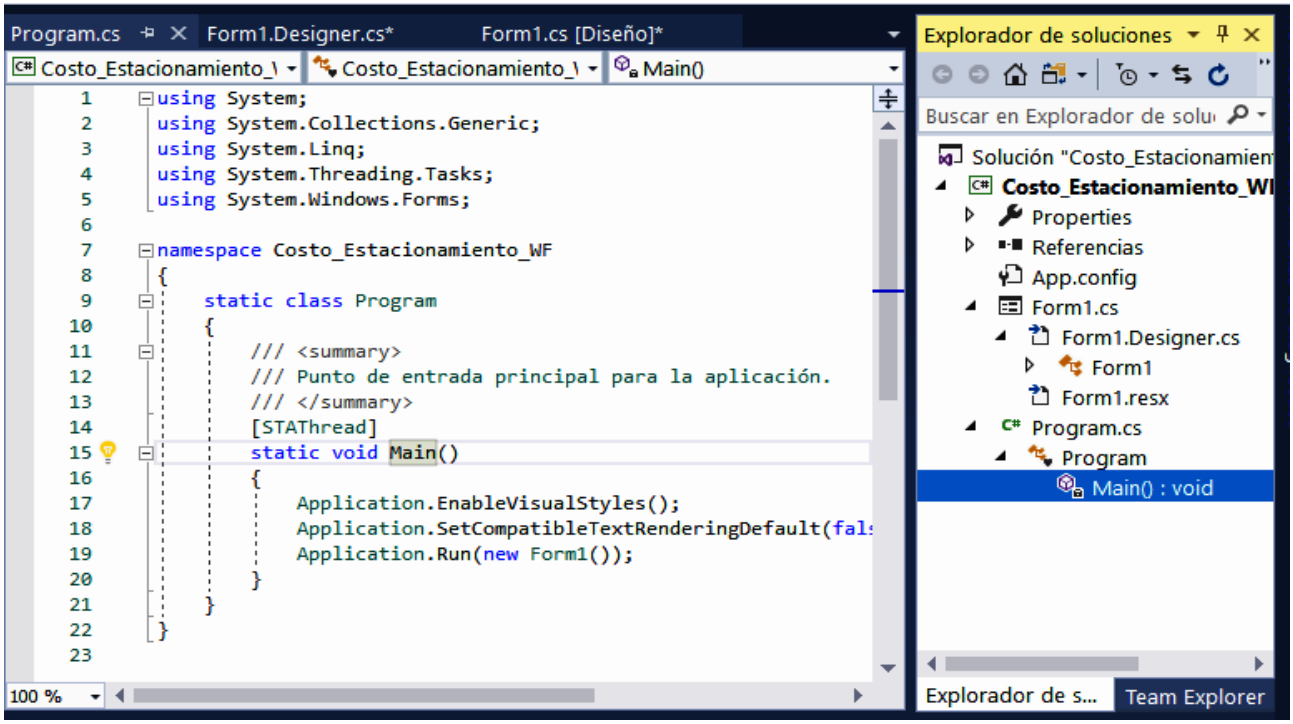
A la **derecha**, por defecto tendremos el ***Explorador de Soluciones*** y debajo la ***Ventana de Propiedades,*** que nos mostrará las variables que pueden modificar las propiedades del Control que se encuentre seleccionado, (por defecto al principio, el **Form1**, y allí, en la propiedad **Text** podemos cambiar el nombre de la Ventana.

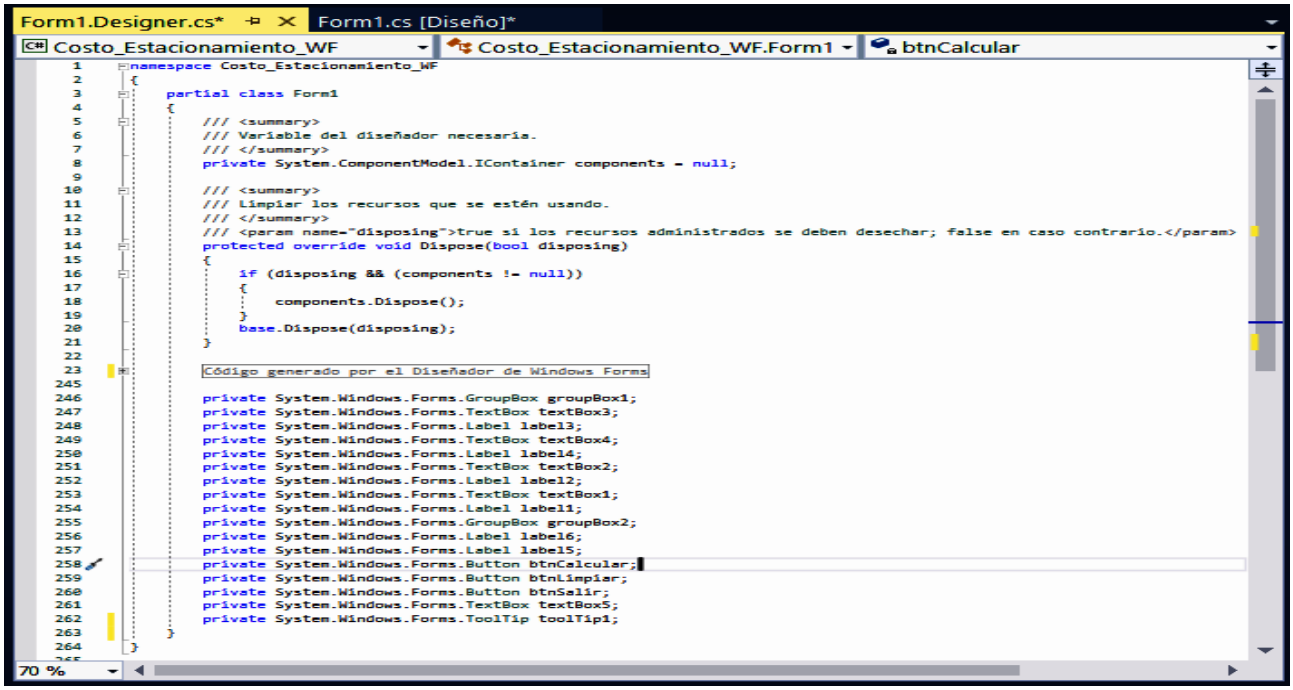
En la ventana de propiedades vemos todas las variables (que podemos cambiar a voluntad) del control seleccionado en el formulario (incluso puede ser el mismo Form1)



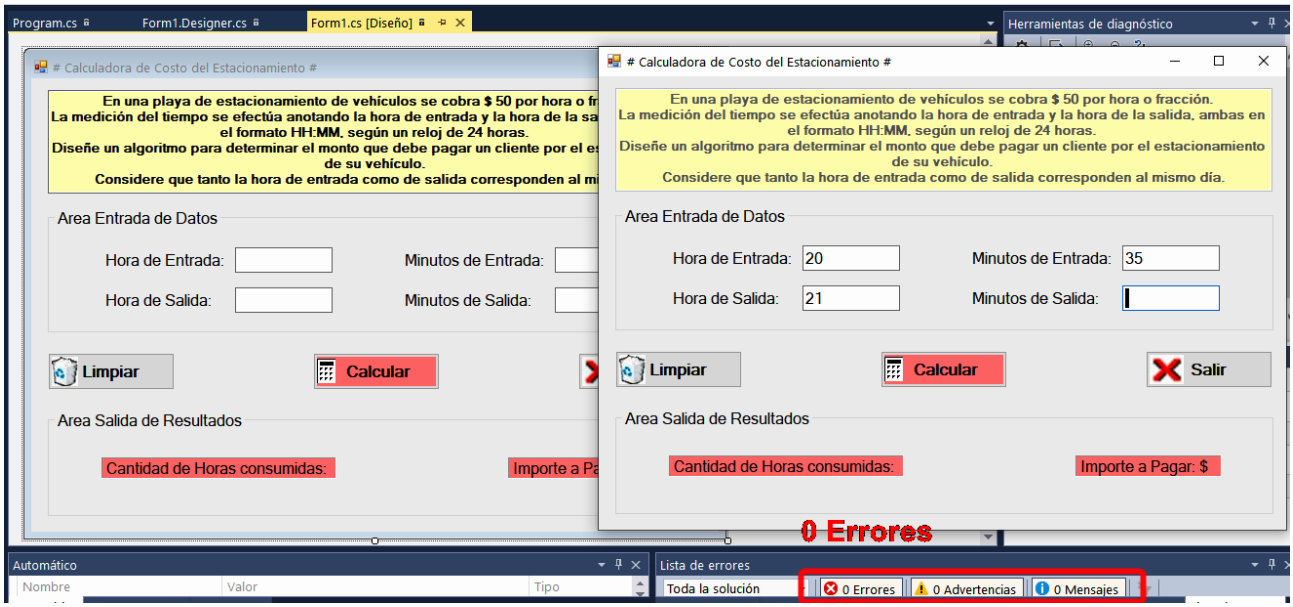
Como puede observarse, se han realizado los cambios que se indican en las ***Propiedades*** de varios de los controles dibujados en el Form1. (Para eso está la ***Ventana Propiedades)***

Aunque aún no hemos escrito una sola línea de código, si vamos al ***Explorador de soluciones***, vemos que ***Program.cs*** ya tiene 23 líneas escritas por Visual Studio C# (VSC#); y en el ***Form1.Designer.cs*** tenemos 264 líneas generadas por (VSC#), y que corresponden fundamentalmente a los controles que hemos colocado en el Form, sus dimensiones y propiedades, etc. Debo decir, que estás líneas de Código, ***SI son editadas*** por el programador, **es a su propio riesgo**, ya que lo más usual es que el programa se inutilice y no pueda ser recuperado.





Ahora bien, NO hemos escrito aun ninguna línea de código, pero, si compilamos, vemos que lo hace con ***"0 Errores"***, el programa se ejecuta, nos permite cargar valores en los TextBox, pero si queremos realizar alguna acción pulsando los botones de comando, NO ocurre NADA, y es Normal que sea así, porque NO hemos escrito aun ninguna línea de programa para ellos. Ese es el proceso que realizaremos a continuación.

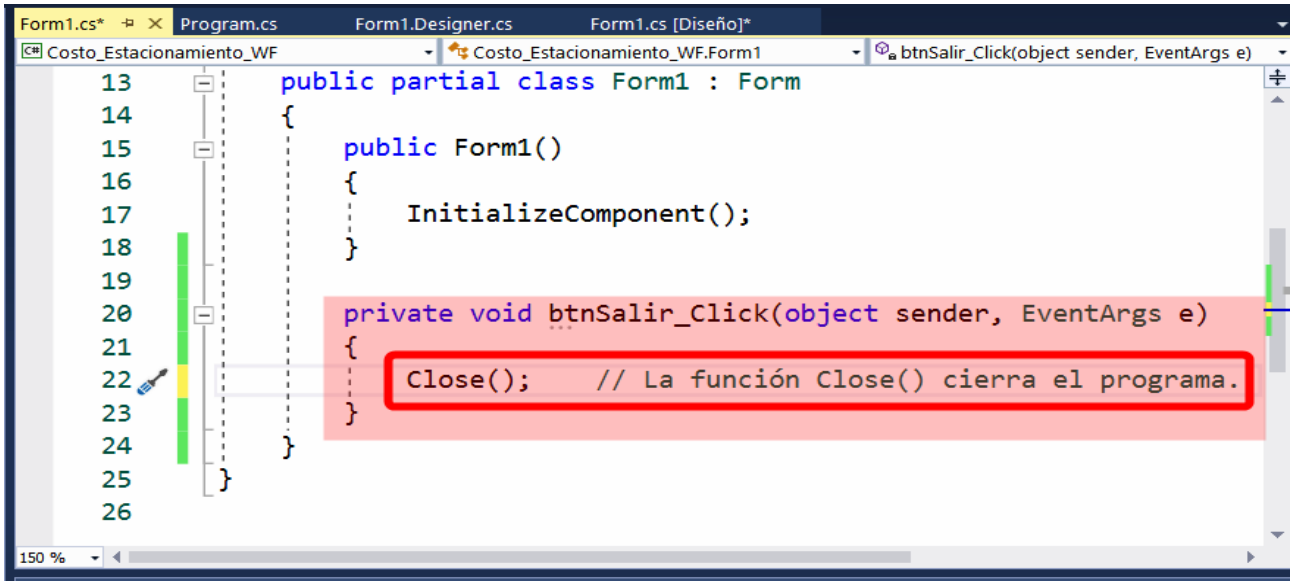


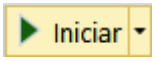
Si estamos en la pestaña de***: "Form1.Designer.cs",*** y hacemos doble clic sobre el "botón Salir", se abre la pestaña: "Form1.cs", con la función ***"btnSalir_Click()"*** enunciada y las llaves de apertura y cierre { …. }. ***btnSalir*** es el Nombre que le hemos dado al botón anterior, y ***_Click*** corresponde al evento o acción del usuario que dispara la ejecución de la función ***"btnSalir_Click()"*** .

Esta función ejecutará todo el código que nosotros agreguemos entre { y }, y allí, nosotros agregamos todo el código que corresponda al algoritmo que deseamos ejecutar. En este caso el trabajo es muy simple, ya que solo queremos salir del programa, y en VSC# hay una función existente que realiza este trabajo, y esta función se llama ***"Close();"*** .

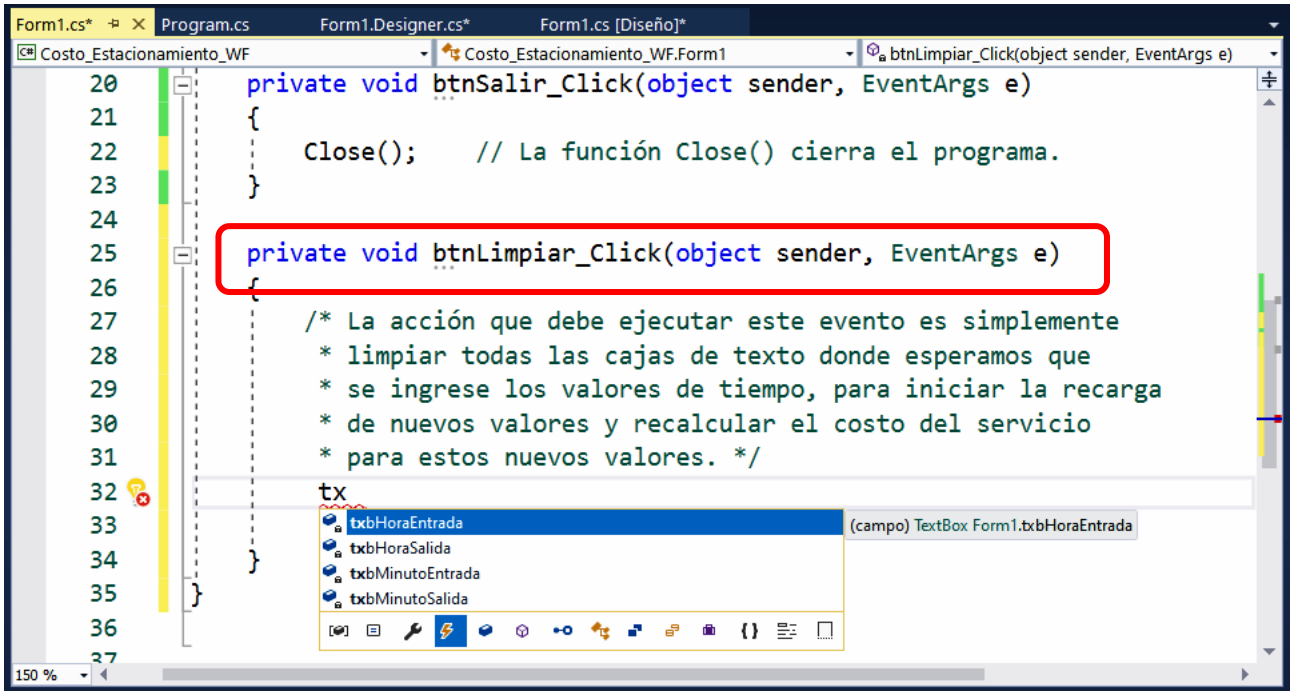
Como toda función, debe incluir los paréntesis () para que incluya los argumentos que deben pasarse a la función para que opere sobre ellos, pero en este caso particular, no necesitamos pasarle ningún argumento a la función.

Además, como toda línea de programa que se precie en VSC#, debe finalizar con la indicación de final de línea de código, que es justamente el ***";".***



Si ahora compilamos: al 

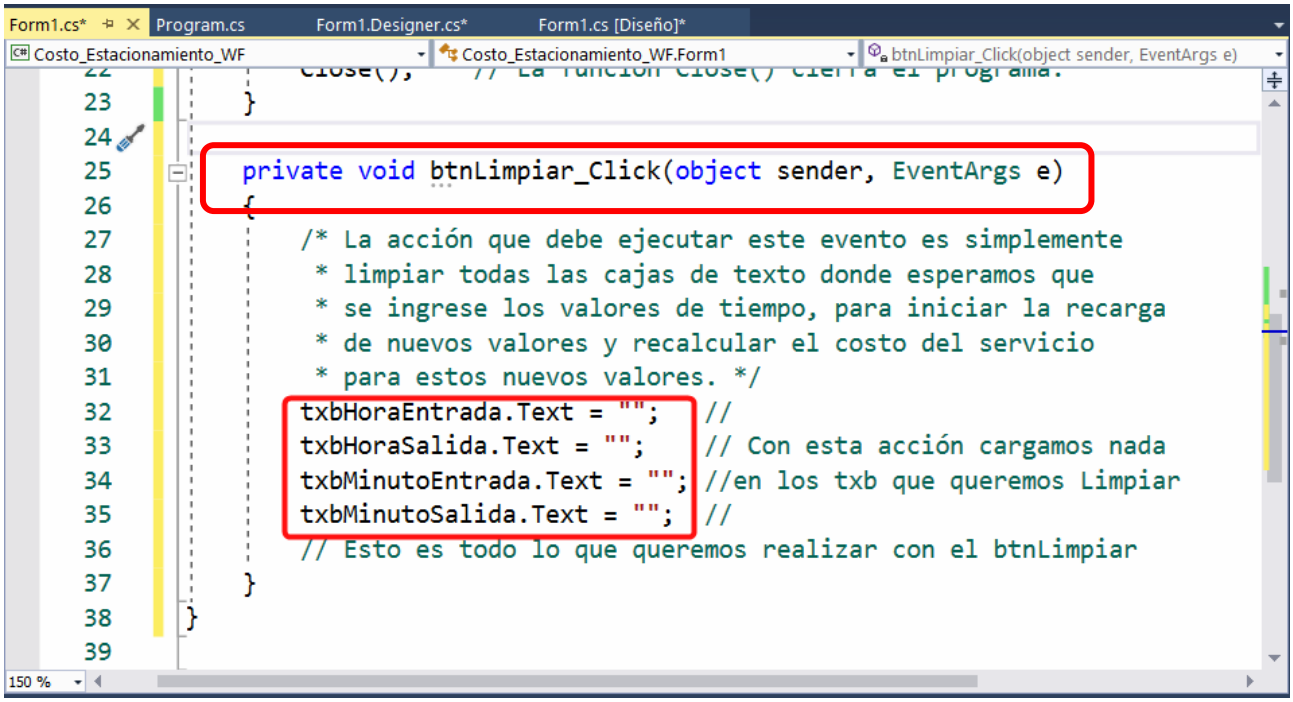
aparecer la ventana de ejecución del programa hacemos **clic** sobre el **btnSalir,** esta acción llama a la función que resaltamos arriba, y veremos que la aplicación se cierra al ejecutarse la función **Close().**

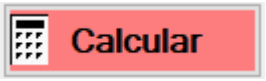


Como vemos, repitiendo el proceso realizado sobre el ***btnSalir***, pero ahora sobre el ***btnLimpiar***, ya nos aparece creada la estructura de la nueva función ***btnLimpiar_Click***, y como queremos limpiar el Texto de los **TextBox** (así como a los **Button** le hemos antepuesto al nombre tres letras

minúsculas que los identifiquen – "**btn**", a los **TextBox,** le anteponemos – "**txb**".), veremos que al comenzar a escribir ***txb,*** no necesitamos más, porque el IDE ya identifica las variables o controles a los que nos queremos referir y nos ofrece la opción de seleccionarlos, luego agregamos **.Text** para indicar que queremos cambiar esa propiedad del control y escribimos el nuevo valor: **""**.

Si ahora compilamos: al aparecer la ventana de ejecución del programa, cargamos valores de HH:MM de Entrada y Salida, y luego hacemos **clic** sobre el **btnLimpiar,** esta acción llama a la función que resaltamos arriba, y veremos que la aplicación limpia los valores ingresados en los cuatro **txb** contenidos en el **GroupBox** del **Área de Entrada de Datos**.

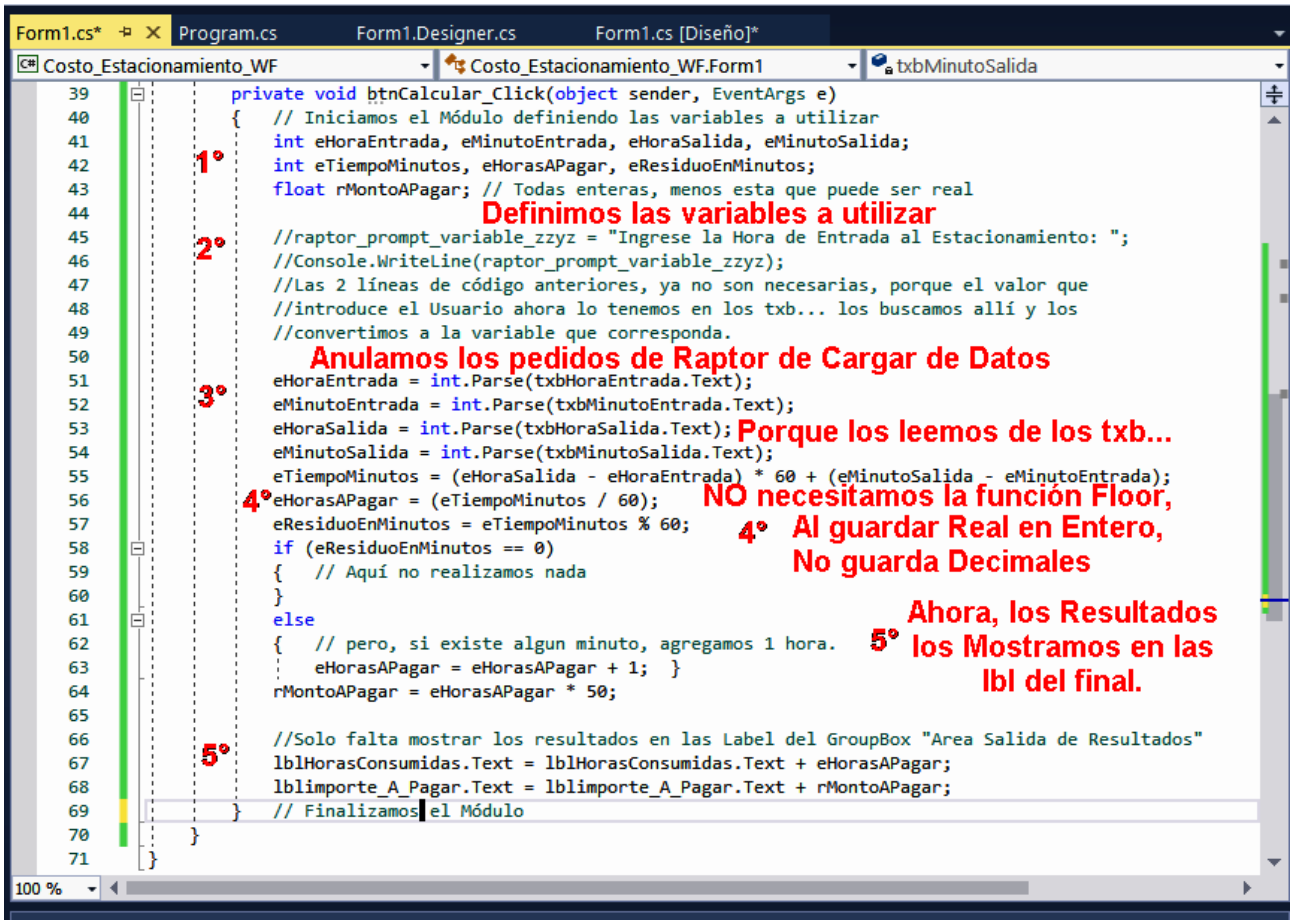




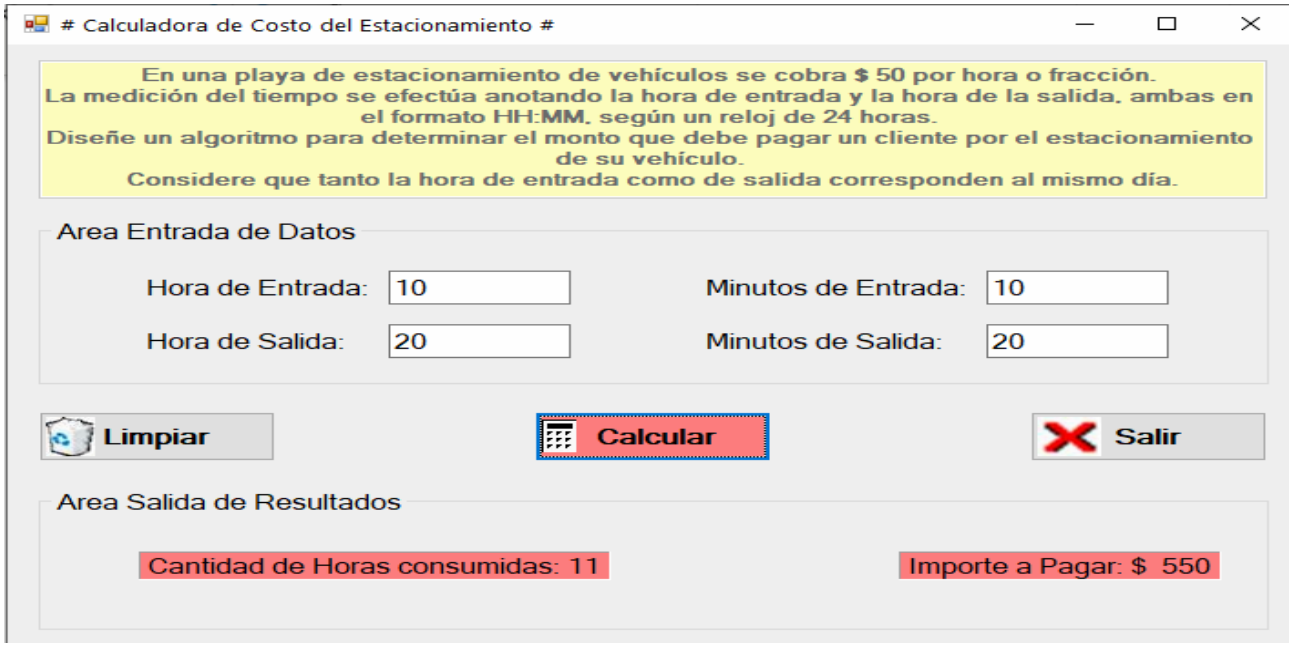
Ya hemos terminado con la programación de dos de los **btn** del programa, pero nos falta el principal, el que realiza el cálculo del importe del ticket, pero esta lógica es la que hemos desarrollado en **Raptor**, solo debemos traer de allí el Código C# generado que corresponde a la pestaña de Cálculo del **"Monto_A_Pagar"**, y debemos definir previamente las variables numéricas que vamos a utilizar (indicando su tipo: Int – para Enteros, y Double (o Float) – para Reales), y realizar las modificaciones correspondientes, sabiendo que ahora los datos los carga el usuario en los **txb**, y los resultados los mostraremos en las **lbl** del final.

Finalmente, tenemos el programa totalmente funcional, ya que hemos cargado el código correspondiente a cada botón de comando.

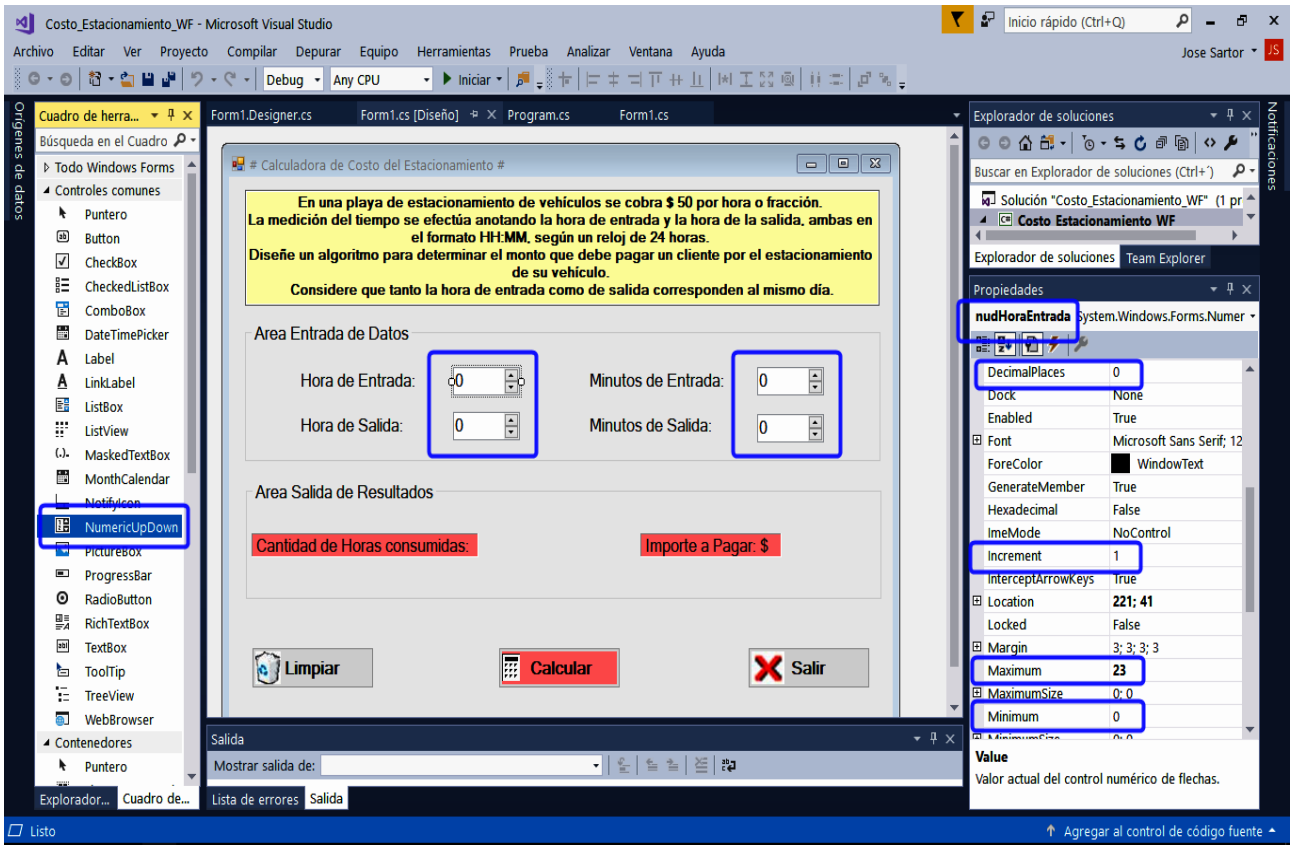
Pero, hay mucho por mejorar todavía. Que pasa si No se introducen números en los txb, o si la hora de salida es anterior a la de entrada. O si no se respeta el valor máximo de 24 y el mínimo de 0 para las horas, o los valores de minutos no están comprendidos entre 0 y 60. Esas cuestiones las resolveremos al ver la *"Guía 7 – Algoritmos REPETITIVOS"*.




```
39      private void btnCalcular_Click(object sender, EventArgs e)
40      {   // Iniciamos el Módulo definiendo las variables a utilizar
41          int eHoraEntrada, eMinutoEntrada, eHoraSalida, eMinutoSalida;
42          int eTiempoMinutos, eHorasAPagar, eResiduoEnMinutos;
43          float rMontoAPagar; // Todas enteras, menos esta que puede ser real
44
45          //raptor_prompt_variable_zzyz = "Ingrese la Hora de Entrada al Estacionamiento: ";
46          //Console.WriteLine(raptor_prompt_variable_zzyz);
47          //Las 2 líneas de código anteriores, ya no son necesarias, porque el valor que
48          //introduce el Usuario ahora lo tenemos en los txb... los buscamos allí y los
49          //convertimos a la variable que corresponda.
50
51          eHoraEntrada = int.Parse(txbHoraEntrada.Text);
52          eMinutoEntrada = int.Parse(txbMinutoEntrada.Text);
53          eHoraSalida = int.Parse(txbHoraSalida.Text);
54          eMinutoSalida = int.Parse(txbMinutoSalida.Text);
55          eTiempoMinutos = (eHoraSalida - eHoraEntrada) * 60 + (eMinutoSalida - eMinutoEntrada);
56          eHorasAPagar = (eTiempoMinutos / 60);
57          eResiduoEnMinutos = eTiempoMinutos % 60;
58          if (eResiduoEnMinutos == 0)
59          {   // Aquí no realizamos nada
60          }
61          else
62          {   // pero, si existe algun minuto, agregamos 1 hora.
63              eHorasAPagar = eHorasAPagar + 1;  }
64          rMontoAPagar = eHorasAPagar * 50;
65
66          //Solo falta mostrar los resultados en las Label del GroupBox "Area Salida de Resultados"
67          lblHorasConsumidas.Text = lblHorasConsumidas.Text + eHorasAPagar;
68          lblimporte_A_Pagar.Text = lblimporte_A_Pagar.Text + rMontoAPagar;
69      }   // Finalizamos el Módulo
70  }
71  }
```



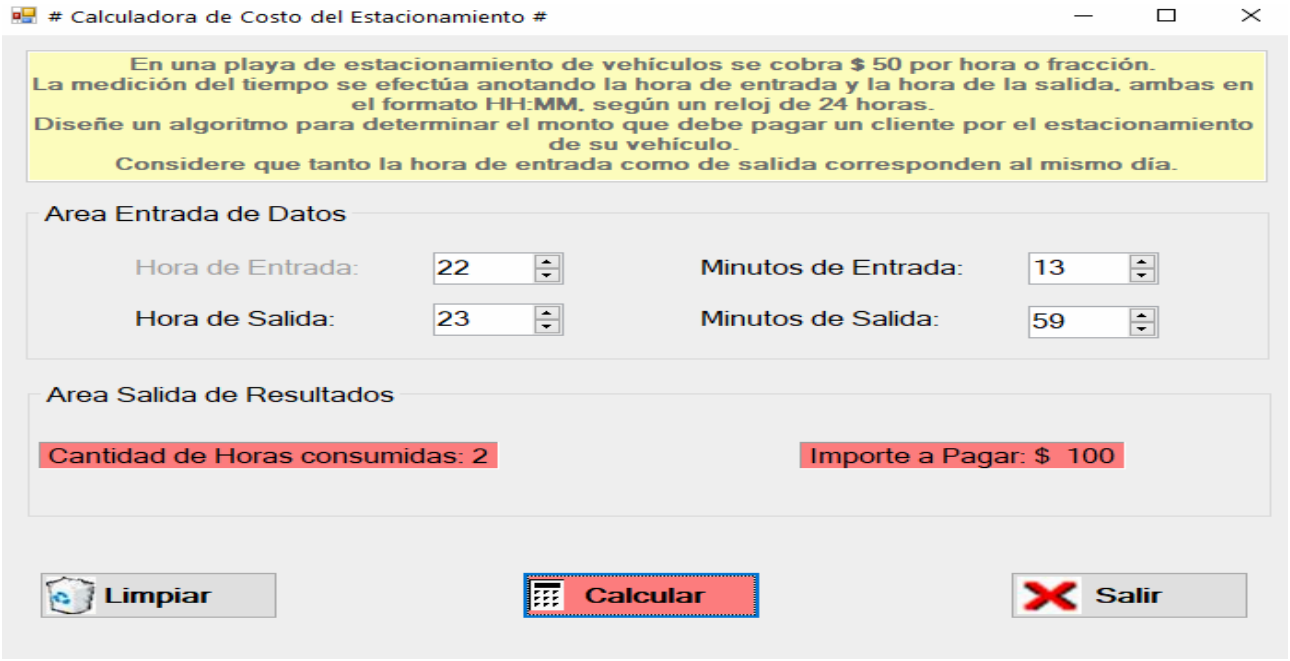
Ahora bien, parte de estos problemas (que el usuario ingrese letras o valores no permitidos en horas o minutos) podemos resolverlos (validando la entrada para que sean solo números y dentro de los valores permitidos -Máximo 23 horas y 60 minutos mínimos 0 para horas y minutos-) con solo el cambio del control "***TextBox***" por un control "***NumericUpDown***", este control retorna números decimales en su propiedad Value, y tiene varias propiedades interesantes que están resaltadas en le imagen siguiente:



A continuación el cambio de código en el ***btnLimpiar*** :

```
25      private void btnLimpiar_Click(object sender, EventArgs e)
26      {
27          /* La acción que debe ejecutar este evento es simplemente limpiar
28           * todas los valores ingrsados por el ususario, para iniciar
29           * la recarga de valores y recalcular el costo del servicio */
30
31          nudHoraEntrada.Value = 0;    //
32          nudMinutoEntrada.Value = 0; // Con esta acción ponemos a cero
33          nudHoraSalida.Value = 0;     //en los nud que queremos Limpiar
34          nudMinutoSalida.Value = 0;  //
35
36          lblHorasConsumidas.Text = "Cantidad de Horas consumidas: ";
37          lblimporte_A_Pagar.Text = "Importe a Pagar: $  ";
38          // Esto es todo lo que queremos realizar con el btnLimpiar
39      }
40
```

Pantalla del programa en ejecución con los controles ***NumericUpDown,*** que solo permiten ingresar valores numéricos dentro de los rangos establecidos en cada control:



En el código del ***btnCalcular*** podemos ver los pocos cambios en el código para adaptarlo al nuevo control, con la importante mejora de que ahora solo se pueden ingresar valores numéricos al control, y los cambios de la función ***Parse*** (que pasaba el ***Texto*** a ***Numero,*** si el texto era un número) a ***Convert*** (que cambia del tipo de Dato N° ***Decimal*** al tipo ***Int***):

```
42
43      private void btnCalcular_Click(object sender, EventArgs e)
44      {   // Iniciamos el Módulo definiendo las variables a utilizar
45          int eHoraEntrada, eMinutoEntrada, eHoraSalida, eMinutoSalida;
46          int eTiempoMinutos, eHorasAPagar, eResiduoEnMinutos;
47          float rMontoAPagar; // Todas enteras, menos esta que puede ser real
48
49          //raptor_prompt_variable_zzyz = "Ingrese la Hora de Entrada al Estacionamiento: ";
50          //Console.WriteLine(raptor_prompt_variable_zzyz);
51          //Las 2 líneas de código anteriores, ya no son necesarias, porque el valor que
52          //introduce el Usuario ahora lo tenemos en los txb... los buscamos allí y los
53          //convertimos a la variable que corresponda.
54
55          eHoraEntrada = Convert.ToInt16(nudHoraEntrada.Value);  //Convierte los N° Decimales
56          eMinutoEntrada = Convert.ToInt16(nudMinutoEntrada.Value);  // retornados por la
57          eHoraSalida = Convert.ToInt16(nudHoraSalida.Value); // propiedad "Value" de los nud
58          eMinutoSalida = Convert.ToInt16(nudMinutoSalida.Value);  // a N° Enteros.
59
60          eTiempoMinutos = (eHoraSalida - eHoraEntrada) * 60 + (eMinutoSalida - eMinutoEntrada);
61          eHorasAPagar = (eTiempoMinutos / 60);
62          eResiduoEnMinutos = eTiempoMinutos % 60;
63          if (eResiduoEnMinutos == 0)
64          {   // Aquí no realizamos nada
65          }
66          else
67          {   // pero, si existe algun minuto, agregamos 1 hora.
68              eHorasAPagar = eHorasAPagar + 1;  }
69          rMontoAPagar = eHorasAPagar * 50;
70
71          //Solo falta mostrar los resultados en las Label del GroupBox "Area Salida de Resultados"
72          lblHorasConsumidas.Text = lblHorasConsumidas.Text + eHorasAPagar;
73          lblimporte_A_Pagar.Text = lblimporte_A_Pagar.Text + rMontoAPagar;
74      }   // Finalizamos el Módulo
75
```