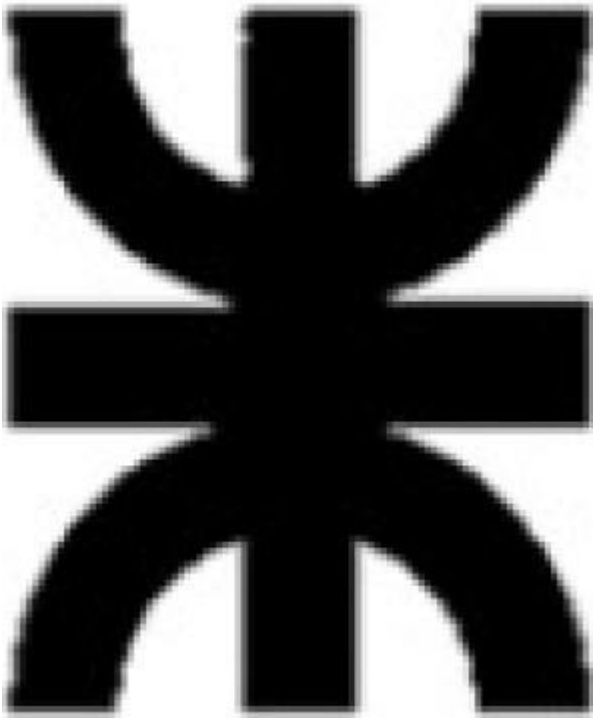


UNIVERSIDAD TECNOLÓGICA NACIONAL
Facultad Regional Reconquista

Programación en Computación
Ciclo Lectivo: 2020

Trabajo Práctico N.º 5

5 – Guía U5 Estructuras SECUENCIALES
Desarrollar los problemas en pseudocódigo, Raptor y Visual C#

GRUPO Nº: 12
INTEGRANTES: Fulano, Mengano, Zutano

Guía Unidad 5:

ALGORITMOS SECUENCIALES

SITUACIONES PROBLEMÁTICAS:

1. **Diseñe un algoritmo que convierta convertir una longitud dada en centímetros a pies.** (Considere que 1 pie = 30.48 centímetros.)
2. **Diseñe un algoritmo para determinar el área de un terreno rectangular en metros cuadrados y en hectáreas considerando que las dimensiones del terreno se dan en metros.**
3. **Dada una cantidad en pesos, obtener la equivalencia en dólares.**
4. **Calcular el nuevo salario de un obrero si obtuvo un incremento del25% sobre su salario anterior.**
5. **Un maestro desea saber qué porcentaje de hombres y que porcentaje de mujeres hay en un grupo de estudiantes.**
6. **Tres personas deciden invertir su dinero para fundar una empresa. Cada una de ellas invierte una cantidad distinta. Obtener el porcentaje que cada uno invierte con respecto a la cantidad total invertida.**
7. **Se desea un programa para calcular el monto bruto, el monto del impuesto y el monto a pagar por la compra de cierta cantidad de unidades de un mismo tipo de producto cuyo costo por unidad (libre de impuestos) es de $ 25.45. La compra está sujeta a un impuesto del 21%.**
8. **Un vendedor recibe un sueldo base más un 10% extra por comisión de sus ventas, el vendedor desea saber cuánto dinero obtendrá por concepto de comisiones por las tres ventas que realiza en el mes y el total que recibirá en el mes tomando en cuenta su sueldo base y comisiones.**
9. **Un alumno desea saber cuál será su calificación final en la materia de Algorítmos.**

   Dicha calificación se compone de los siguientes porcentajes:
   55% del promedio de sus tres calificaciones parciales.
   30% de la calificación del examen final.
   15% de la calificación de un trabajo final.
10. **Diseñe un algoritmo que convierta convertir una longitud dada en centímetros a pulgadas.** (Considere que 1 pulgada = 2.54 centímetros.)
11. **Realizar un algoritmo que lea una temperatura en °F y realice la conversión a °K y °C.**
12. **Ingresar una temperatura en °C (Centígrados o Celsius), mostrar sus valores equivalentes en °K (Kelvin) y en °F (Fahrenheit).** ***(Problema desarrollado como ejemplo)***

ACTIVIDADES:
Resolver las situaciones problemáticas anteriores, mediante Pseudocódigo, Diagrama de Flujo en RAPTOR y codificación en C#. *(Salida por Consola)*

Conocimientos necesarios:

U 2) Algoritmos. U 3) Tipos de Datos. U 4) Diagramación lógica. Diagramas de Flujo y U 5) Estructuras Secuenciales. Uso de Asignaciones, Entradas y Salidas de Datos. Resolución de problemas usando estructuras secuenciales para la formulación de sus algoritmos.

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 4.

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

Resumen de estos:

El ***Diseño del Algoritmo,*** *[Secuencia ordenada de pasos - Sin Ambigüedades – que conducen a la solución de un Problema].*
Que además debe ser:
i - Preciso ***ii – Definido*** ***iii – Finito.***

Esto implica identificar las tareas más importantes y ponerlas en el orden en que deben ejecutarse. Estos pasos pueden repetirse ***(Refinamiento Progresivo, Diseño Descendente o Top-Down)*** hasta obtener un ***Algoritmo:***

Claro, ***Preciso,*** ***Completo.***

Además, un Algoritmo consta de tres partes principales:
i - Entrada o información necesaria para la ejecución del **Algoritmo.**
ii - Proceso u operaciones necesarias **para la Solución.**
iii - Salida o respuestas dadas por el **Algoritmo.**

NOTA: Los errores más comunes en programación son:

1. ***De Sintaxis (mucho más comunes),*** Errores en las palabras o identificadores permitidos o en los signos de puntación.
2. ***De Lógica,*** *Se* pensó mal la forma de resolver el problema.
3. ***En tiempo de Ejecución.*** Cuando el programa ya se está ejecutando, y se interrumpe la ejecución por algún error del Programa. *(División por cero, o la raíz cuadrada de un número negativo)*

Problema 12:

Ingresar una temperatura en °C (Centígrados o Celsius), mostrar sus valores equivalentes en °K (Kelvin) y en °F (Fahrenheit). (Problema desarrollado como ejemplo)

1) Pseudo-Código:

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

algoritmo <*nombre identificador del algoritmo*>

¡Comentarios! **// declaraciones y sentencias no ejecutables.**
// Fecha, Programador, etc.

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

Real Celsius, Kelvin

Inicio

// acciones, sentencias ejecutables ¡Comentarios!

Leer (grado_Celsius) **// carga Datos**

// realiza el proceso de Cálculo

Asignar grado_Kelvin ← grado_Celsius + 273,15

Asignar: grado_Fahrenheit ← (grado_Celsius * 9 / 5) + 32

Imprimir: (“°K = “ grado_Kelvin, “ °F = “ grado_Fahrenheit)

// muestra o imprime el resultado

Fin

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.
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.

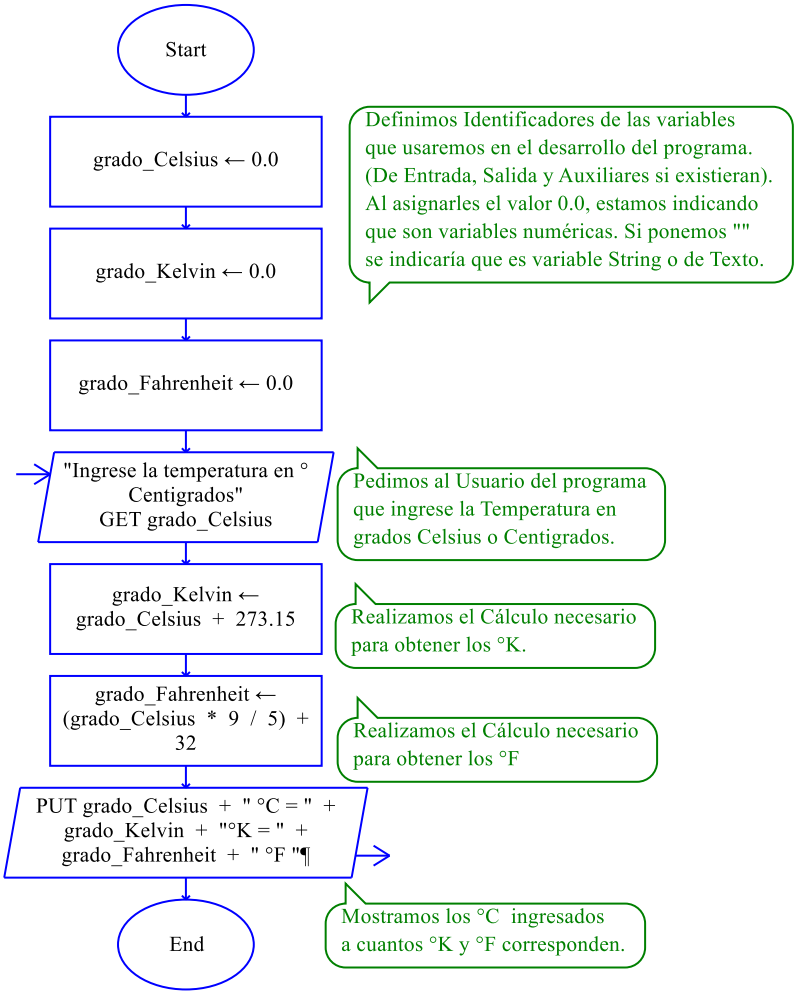
Utiliza **símbolos normalizados** unidos por flechas

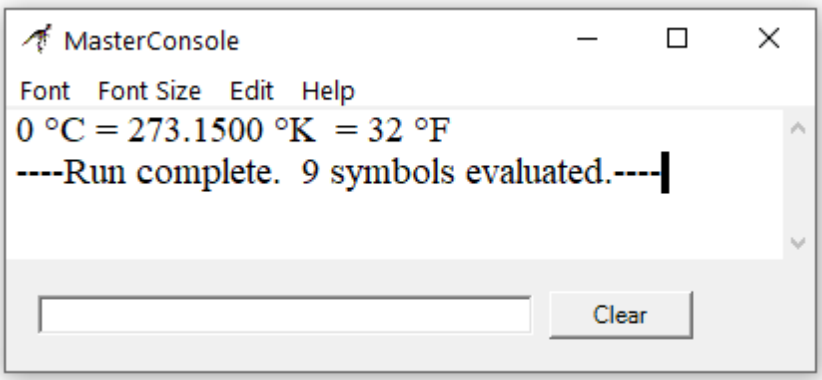
(**líneas de flujo**) que indican el orden en que debe

ejecutarse cada paso.

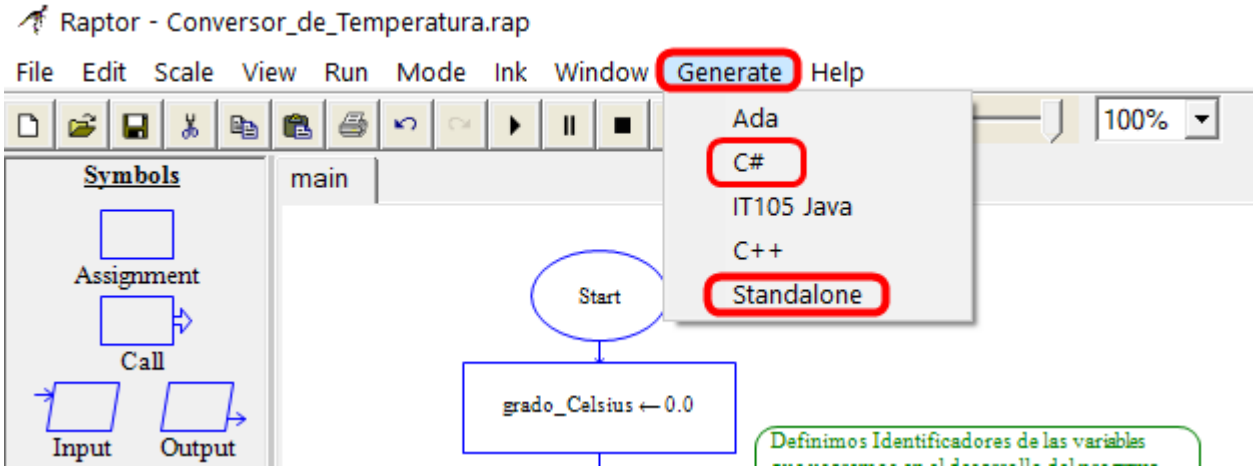
Es muy importante porque permite:

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


Start
grado_Celsius ← 0.0
grado_Kelvin ← 0.0
grado_Fahrenheit ← 0.0
"Ingrese la temperatura en ° Centigrados"
GET grado_Celsius
grado_Kelvin ← grado_Celsius + 273.15
grado_Fahrenheit ← (grado_Celsius * 9 / 5) + 32
PUT grado_Celsius + " °C = " + grado_Kelvin + "°K = " + grado_Fahrenheit + " °F "¶
End
Definimos Identificadores de las variables que usaremos en el desarrollo del programa. (De Entrada, Salida y Auxiliares si existieran). Al asignarles el valor 0.0, estamos indicando que son variables numéricas. Si ponemos "" se indicaría que es variable String o de Texto.
Pedimos al Usuario del programa que ingrese la Temperatura en grados Celsius o Centigrados.
Realizamos el Cálculo necesario para obtener los °K.
Realizamos el Cálculo necesario para obtener los °F
Mostramos los °C ingresados a cuantos °K y °F corresponden.


MasterConsole
Font Font Size Edit Help
0 °C = 273.1500 °K = 32 °F
----Run complete. 9 symbols evaluated.----
Clear

Como se puede ver, hemos realizado el Diagrama de Flujo (D.F.) en Raptor.
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.



Pero en Raptor podemos ver en la línea de Menu, que existe una carpeta llamada ***Generate***, y dentro de ella, hemos remarcado con recuadro Rojo dos opciones que utilizaremos: 1) ***Standalone*** y 2) ***C#.***

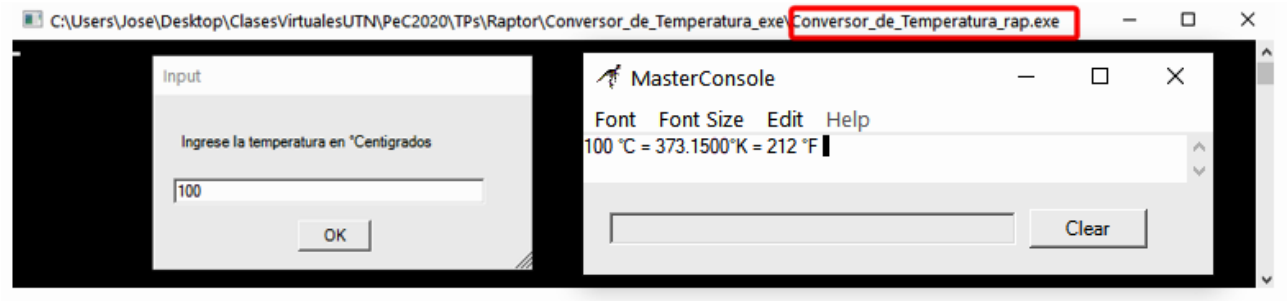
1) Si hacemos clic en ***Standalone*** Raptor nos informa que en la Carpeta (Raptor) desde donde estamos ejecutando el programa ha creado una carpeta que tiene el mismo nombre del programa en ejecución, pero le ha agregado un _exe, o sea ***"Conversor_de_Temperatura_exe"***

Dentro de esa carpeta, tenemos un archivo llamado:

"Conversor_de_Temperatura_rap.exe"

Como la extensión de su nombre lo indica ***".exe",*** este es un archivo que se autoejecutará con un doble clic, y podremos ejecutar el programa sin necesidad de tener Raptor en la PC, pero son necesarios todos los archivos contenidos en la carpeta.

Si hacemos doble clic sobre el archivo referenciado, obtenemos la siguiente salida:



Los archivos son: **DLL** (sigla en inglés de dinamic-link library) es el término con el que se refiere a los archivos con código ejecutable que se cargan bajo demanda de un programa por parte del sistema operativo.

Esta denominación es exclusiva a los sistemas operativos Windows siendo «.dll» la extensión con la que se identifican estos ficheros.

Mas información sobre archivos DLL en:

https://support.microsoft.com/es-es/help/815065/what-is-a-dll

2) Si hacemos clic en ***C#,*** Raptor genera un archivo con el código de nuestro programa en C# llamado "***Conversor_de_Temperatura.cs",*** y lo abrirá en el ***Bloc de Notas*** de Windows.

   La extensión CS hace referencia al lenguaje CSharp o C# (NO exactamente el Microsoft Visual C#, sino el Mono C#), pero que, con pocas adaptaciones lo podremos ejecutar en el Visual C#.

Lo que es importante ver y destacar de este programa generado por Raptor, es que podemos comenzar a aproximarnos a la estructura y sintaxis del lenguaje C.

Están resaltados con colores los elementos fundamentales: el uso de las llaves para marcar el comienzo y fin de los bloques de código, y el uso de los " **;** " para indicar los finales de cada línea de código.

Ver la copia del código generado por Raptor en la imagen siguiente:

*Conversor_de_Temperatura.cs: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
using System;
using System.IO;
using System.Text;

namespace Conversor_de_Temperatura
{
   public class main_class
   {
      static System.Random random_generator = new System.Random();
      public static void Main(string[] args)
      {
         string raptor_prompt_variable_zzyz;
         ?? grado_fahrenheit;
         ?? grado_celsius;
         ?? grado_kelvin;

         grado_Celsius =0;
         grado_Kelvin =0;
         grado_Fahrenheit =0;
         raptor_prompt_variable_zzyz ="Ingrese la temperatura en °Centigrados";
         Console.WriteLine(raptor_prompt_variable_zzyz);
         grado_Celsius= Double.Parse(Console.ReadLine());
         grado_Kelvin =grado_Celsius+273,15;
         grado_Fahrenheit =(grado_Celsius*9/5)+32;
         Console.WriteLine(grado_Celsius+" °C = "+grado_Kelvin+" °K = "+grado_Fahrenheit+" °F ");
      }
   }
}
```

El archivo comienza con los ***using***, que se usa para cargar los núcleos que necesitará el programa para ejecutarse. Aquí usa ***System*** (del Sistema), ***System.IO*** [las Entradas (INPUT) y Salidas (OUTPUT) del sistema] y por último ***System.Text*** (para el manejo de Textos); pero no debemos ocuparnos de esto porque el Visual C# cargará los suyos propios.

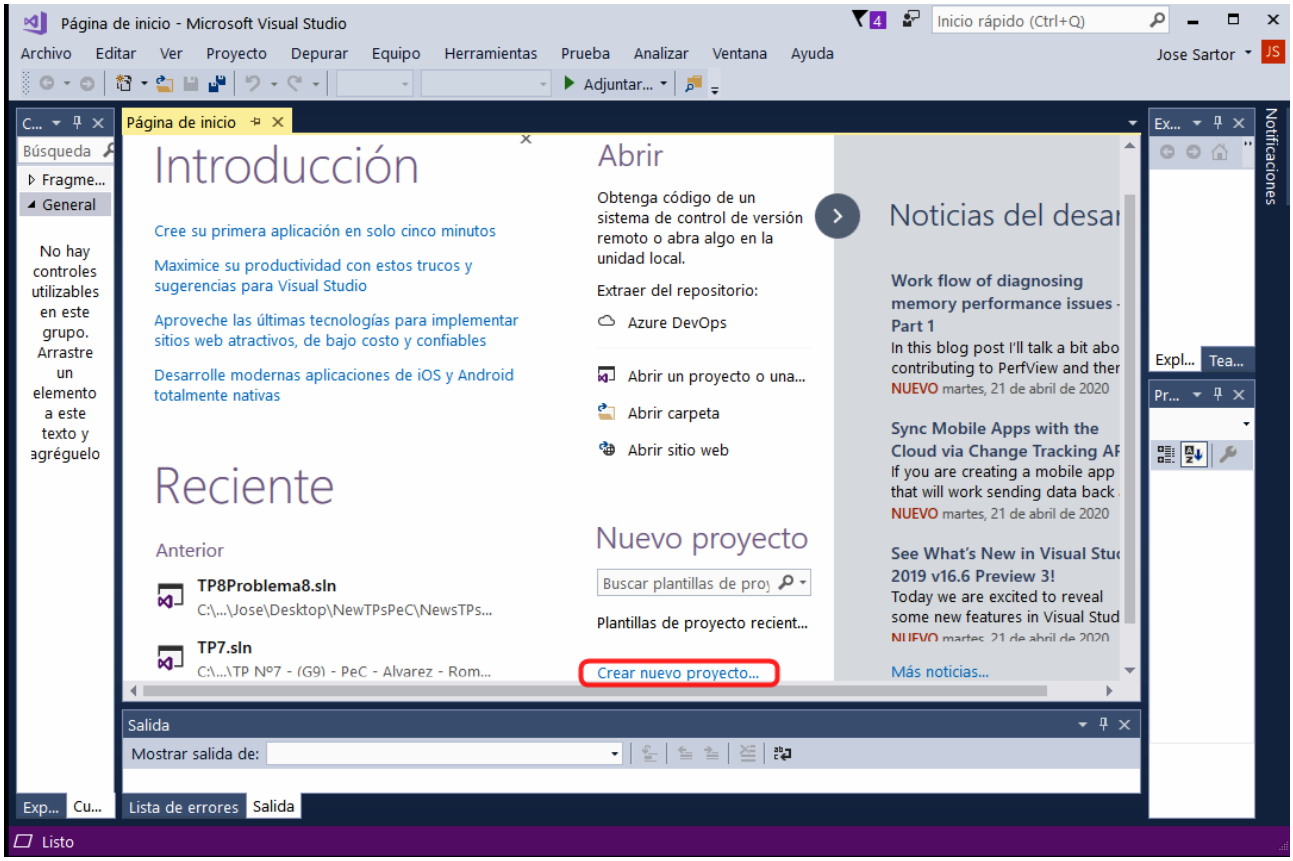
Luego crea un ***namespace Conversor_de_Temperatura***, que es el *Espacio de Nombres para nuestro Objeto*, el programa "*Conversor_de_Temperatura*", luego genera una *Clase Pública llamada clase principal* ***"public class main_class".***

A continuación crea la primera y única función obligatoria en todo programa en C, la función *Main(),* que es pública y que es una función rara porque no devuelve nada *(o sea void = vacio):* ***"public static void Main(string[] args)",*** y entre () podemos ver los argumentos que se pasan a la función.

El bloque de código que nosotros usaremos en Visual C# es el que se encuentra entre las llaves resaltadas con celeste, después de las cuatro primeras líneas.

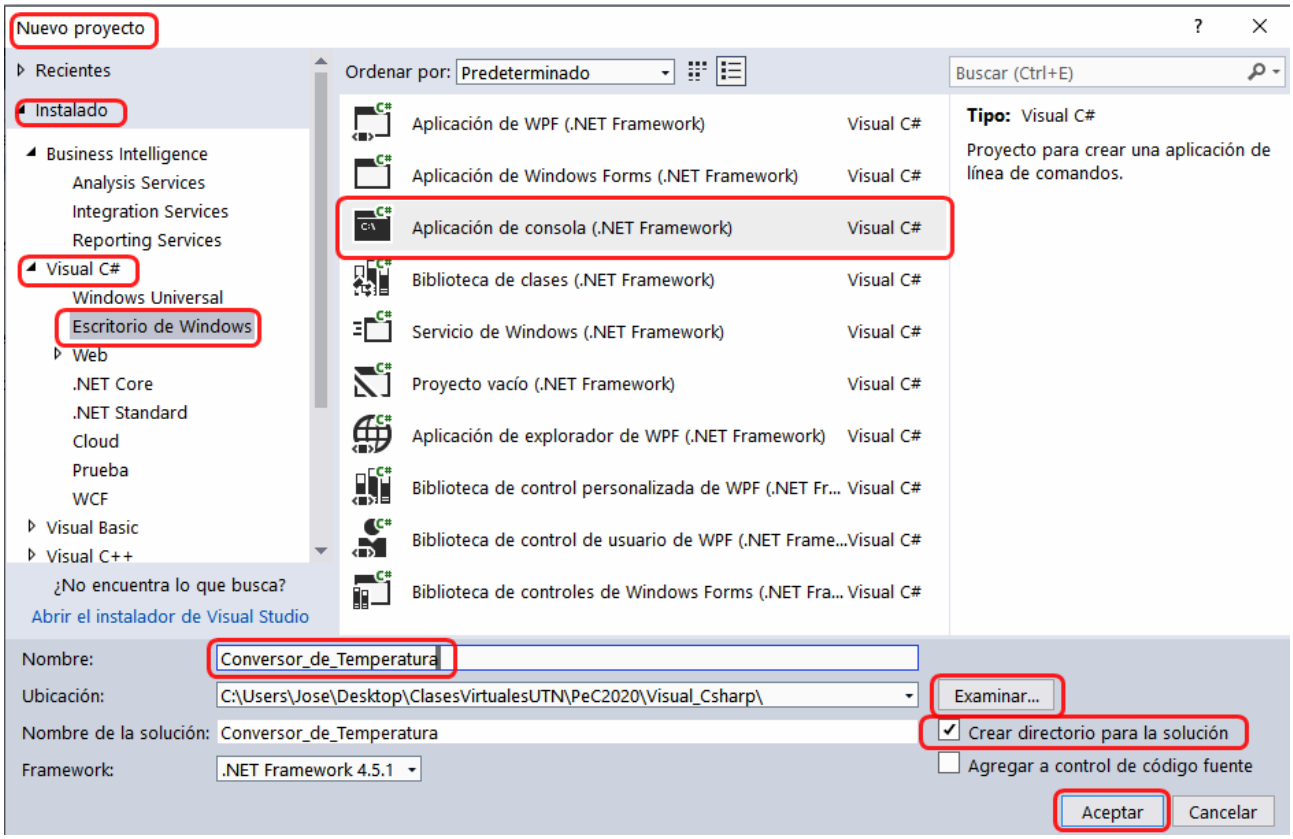
Ahora vamos a realizar este mismo programa en Visual C#, espero ya lo tengan instalado, de lo contrario, en Moodle tienen el enlace para descargarlo:
https://frrq.cvg.utn.edu.ar/mod/url/view.php?id=5916&redirect=1
y allí deben descargar e instalar el ***"Visual Studio Community 2017"***

Página de inicio:



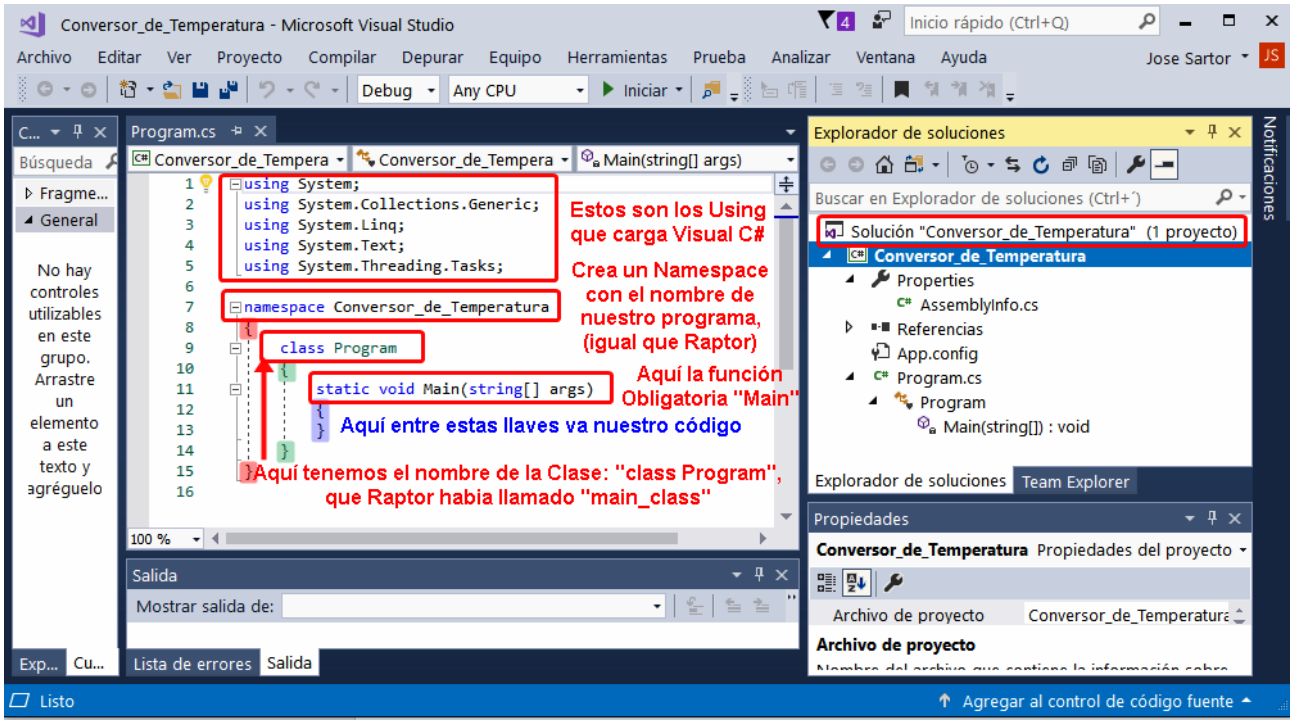
Hacemos clic en ***Crear nuevo proyecto…*** y se abre la ventana ***"Nuevo proyecto"***, allí debemos elegir entre los lenguajes instalados el ***"Visual C#",*** y aquí seleccionamos crear una aplicación para ***"Escritorio de Windows",*** y dentro de las opciones que nos aparecen elegimos ***"Aplicación de consola (.NET FrameWork)"***

Abajo, en la ventana, en ***"Nombre"*** de la aplicación, cambiamos el nombre por defecto que aparece *("ConsoleApp1")* por el mismo nombre de la aplicación que hemos creado en Raptor: "***Conversor_de_Temperatura***"

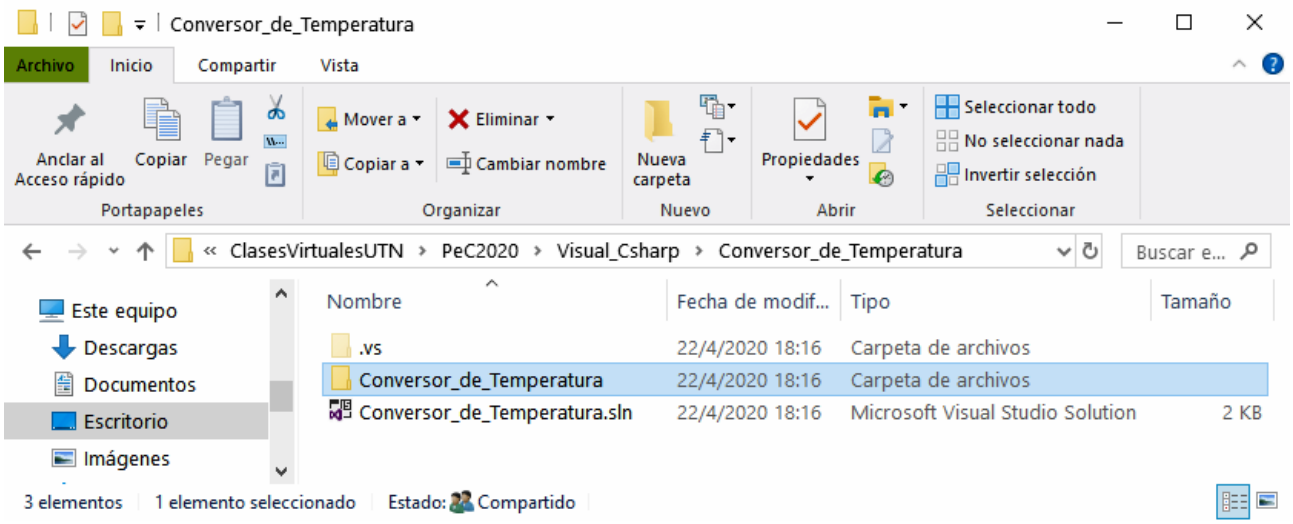


Con el botón ***"Examinar…"*** se abre el explorador de Windows, y allí elegimos el Directorio donde queremos la carpeta con todos los archivos de la Solución.
Debemos asegurarnos de tildar la opción ***"Crear directorio para la solución",*** y finalmente pulsamos el botón ***"Aceptar"***

Aparecerá una nueva Ventana con el Título: "***Conversor_de_Temperatura – Microsoft Visual Studio***", y aquí ya estamos en el **Entorno de Desarrollo Integrado** (IDE - Integrated Development Enviroment) de ***Microsoft Visual Studio***.
Un entorno de desarrollo integrado es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (GUI), que hacen que sea mucho más fácil trabajar en programación que escribiendo código en el Bloc de Notas.
Los IDE proveen un marco de trabajo amigable para la mayoría de los lenguajes de programación.



Todo este código que aparece en la Ventana ***"Program.cs"*** fue creado automáticamente por el IDE de Visual. (En Rojo comentamos las diferencias con el código creado por Raptor). A la derecha tenemos la ventana del ***"Explorador de Soluciones",*** que nos informa que ha creado un Proyecto llamado "***Conversor_de_Temperatura***", y con el Explorador de Windows podemos ver que aparece en el Directorio que le hemos indicado, y ya tiene carpetas y archivos creados:



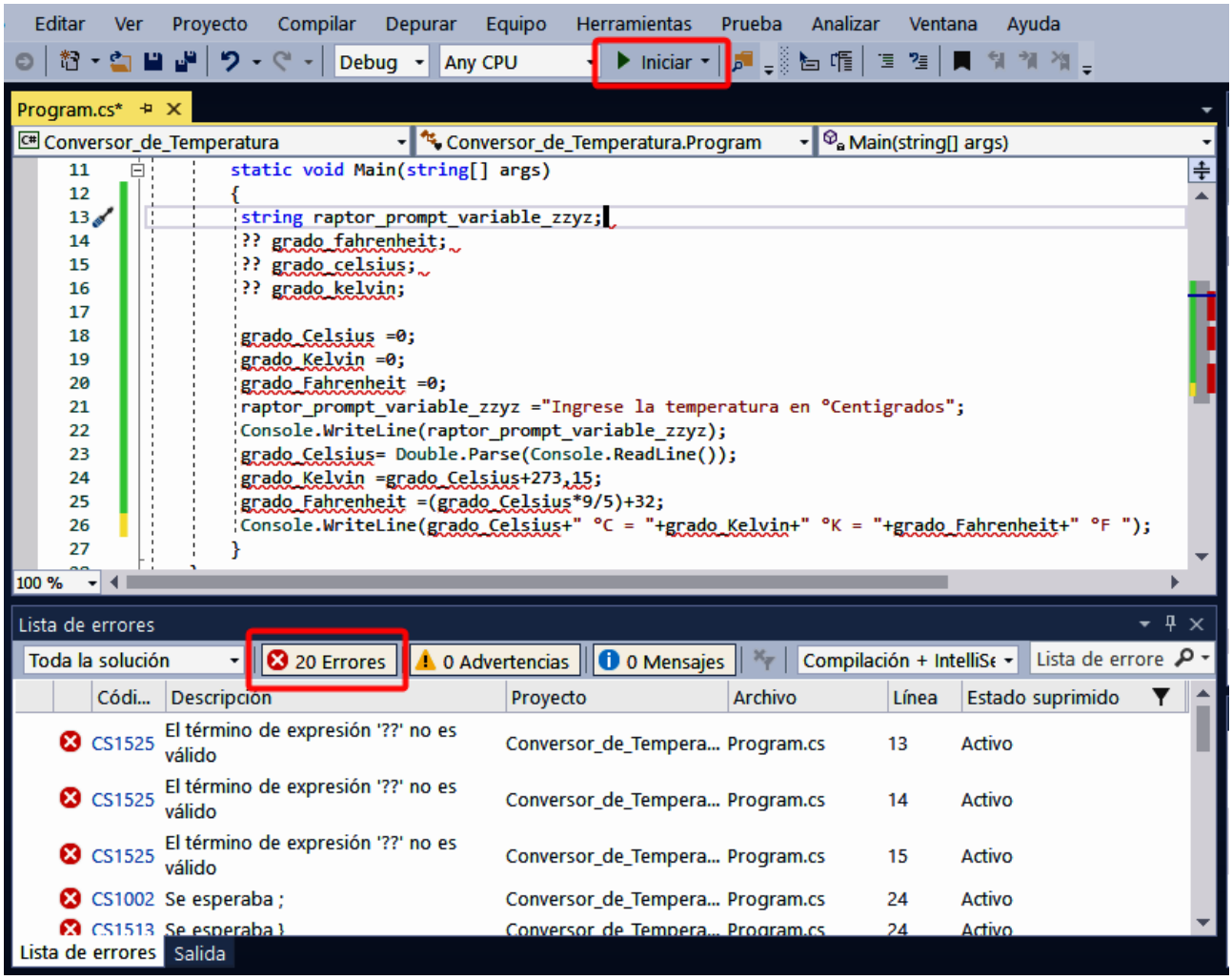
Ahora deberíamos comenzar a cargar el Código para decirle al programa lo que queremos que haga, y aquí insertaremos la parte del código en C# que nos ha generado Raptor dentro del Módulo de la función "***Main***".

Como se ve, han aparecido un montón de subrayados rojos en ondas que indican que al Visual C# NO le gustan (o, mejor dicho, NO las entiende)

Si compilamos el programa haciendo clic en el botón ***Iniciar***, el programa No se compila y en la Ventana ***Lista de errores,*** nos indica que la solución tiene 20 Errores.

Si nos fijamos en los 3 primeros errores, nos dice que ***"??"*** no es válido. En Lenguaje C y en C#, todo lo que continúa a ***"//"*** se considera un comentario, (o sea, no existen para el compilador).

Si antecedemos las líneas 13, 14 y 15 con ***//***, estas líneas toman color verde y los errores ya son 14.



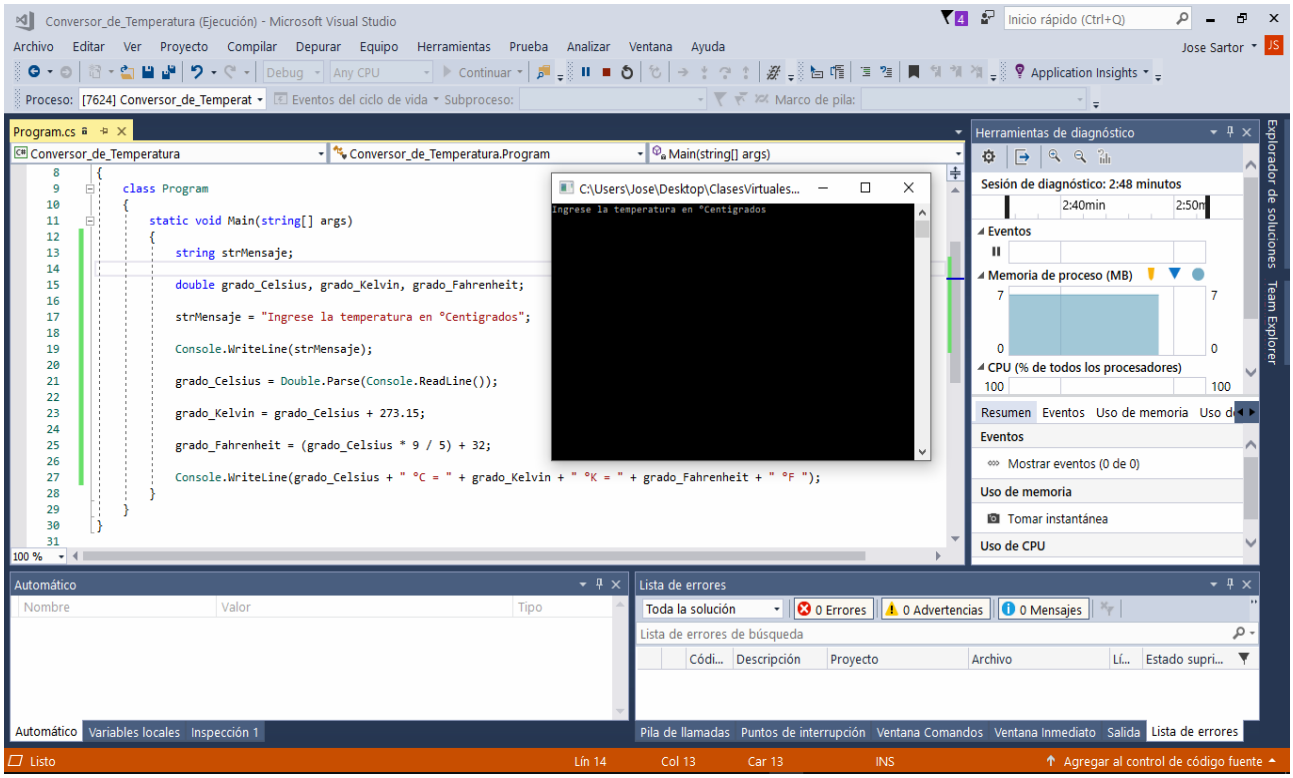
Esto nos indica que esas líneas las podemos quitar y no pasara nada.

Luego, las líneas 18, 19 y 20 también aparecen subrayadas en rojo, indicando que C# no sabe lo que significan, y si vamos a los errores reportados para estas líneas nos indica que: *"El nombre xxxx no existe en el contexto actual"*

C# es un lenguaje fuertemente ***Tipeado*** (implica que antes de usar una variable, debo decirle al compilador que ese identificador es una variable e informarle de que ***tipo*** es.

Si escribimos `double` (luego veremos los tipos de datos válidos) delante de estas 3 líneas, ahora solo nos quedan 3 errores en el programa, y si miramos el código, vemos que hay un subrayado rojo solo en la línea: ***"grado_Kelvin = grado_Celsius + 273,15;",*** y específicamente después de la coma de la constante ***273,15;,*** y este es un error muy común, debido a la configuración regional de los teclados y los S.O. Simplemente cambiemos la coma por el punto (que se usa como separador de decimales en Ingles) y veremos que ya NO tenemos ***Errores, Advertencias o Mensajes*** en nuestro código.

Ahora con clic en ***Iniciar*** veremos nuestro 1° programa en Visual C# corriendo.



El programa se está ejecutando en el modo consola del S.O. Windows (Ventana Negra con letras blancas).

En el programa además de las variables del tipo `double` (numérica con decimales y de doble precisión) hay definida otra variable del tipo `string`, (o cadena de Texto) y con un nombre que podemos cambiar.

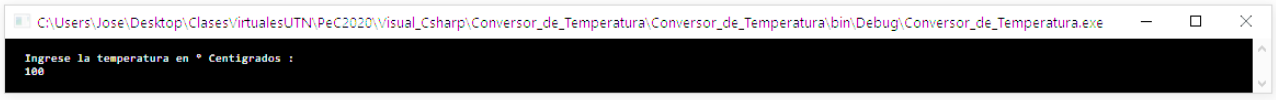
Aprovechemos aquí para cambiar el identificador de esta variable:

`string raptor_prompt_variable_zzyz; por otro: string strMensaje;`

Al intentar hacer esto, nos saldrá una ventana que nos indica que: *"no se permiten cambios mientras se ejecuta código"* , no nos olvidemos que nuestro programa estaba corriendo, pero a la derecha del botón iniciar tenemos botones parecidos a los de un grabador, y pulsamos el de pausa "||" o el de detener "■", y allí podremos cambiar la variable; esta aparecerá subrayada en verde, y si estacionamos el cursor del ratón allí, se abre una ventana que indica que: La variable '`strMensaje`' se ha declarado pero nunca se usa. Además, notaremos que aparecen dos líneas nuevas con subrayado rojo, y son exactamente aquellas que usan el nombre de variable que hemos cambiado, y aquí también deberemos cambiar el nombre de la variable existente por la nueva.

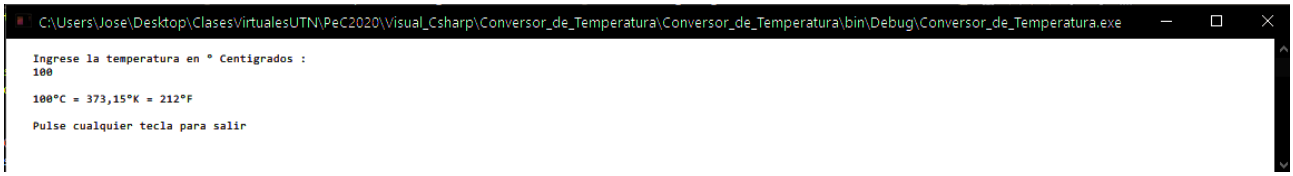
Si ahora corremos nuevamente el programa, veremos en la consola que nos pide:

```
Ingrese la temperatura en ° Centigrados:
```

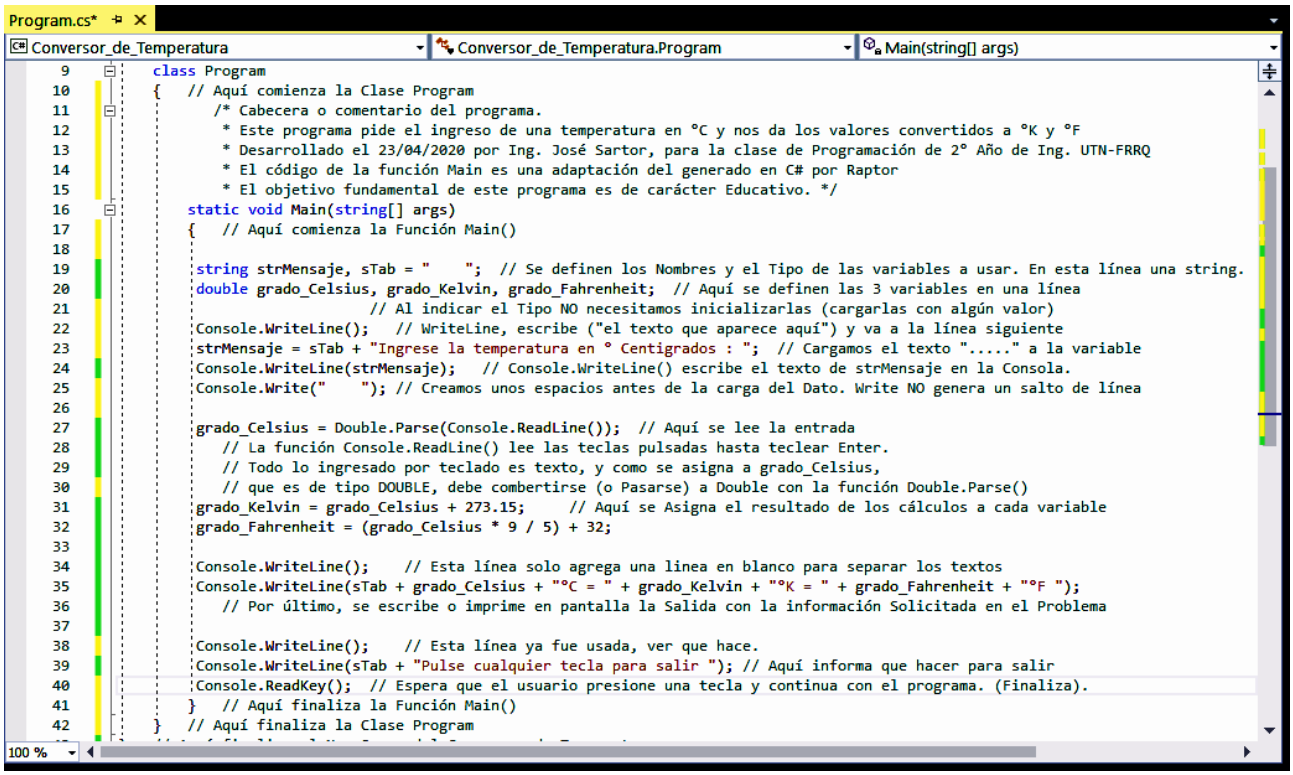


Ingresamos la temperatura deseada, pero al pulsar la tecla *Enter*, no logramos ver nada más. Esto es debido a la velocidad de proceso de la PC, ya que una ves que lee el valor del teclado, lo convierte a *double*, hace los cálculos y los asigna a las variables respectivas, que después muestra en la consola y finalmente termina la ejecución del programa y cierra la consola; pero todo esto se realiza a tal velocidad, que nuestros ojos NO se enteran.

Para que esto NO ocurra y mejorar la presentación he agregado algunas líneas, y la salida final en consola es la siguiente:



El código final del programa (con comentarios agregados – las líneas verdes) queda según se aprecia en la siguiente imagen:



Como ven, este programa está cargado de líneas verdes (comentarios), y es auto explicativo; esta es justamente la forma conveniente de trabajar en programación, o sea, que el programa se vaya auto documentando a medida que se construye, y cuando lo queramos modificar en el futuro, sepamos que es lo que hicimos en su momento.

Quiero recordar que, *(dejando de lado las correcciones realizadas en la definición de variables, y en el problema del remplazo de la coma decimal por el punto)* la lógica o algoritmo del programa en Visual C# es el que hemos generado en Raptor para C#.

Tipos de Datos más usados en C# - (*Tipos básicos o internos)*

Los tipos básicos como hemos dicho son espacios predefinidos y categorizados donde se almacena información. C# tiene una ventaja y característica especial sobre los demás lenguajes de programación modernos y es que cada vez que se crea un objeto de un tipo básico, éstos son mapeados internamente a un tipo primitivo de la plataforma .NET el cual es parte del CLS (Especificación común del lenguaje) lo cual nos permite acceder y hacer uso de estos desde cualquier lenguaje de la plataforma .NET. Es decir, si es que creamos un objeto de tipo int (entero) en C#, ese objeto podrá ser usado como tal dentro de J#, JScript, Visual Basic .NET y cualquier otro lenguaje que conforme los requisitos de .NET.

La plataforma .NET de Microsoft es un componente de software que puede ser añadido al sistema operativo Windows. Provee un extenso conjunto de soluciones predefinidas para necesidades generales de la programación de aplicaciones, y administra la ejecución de los programas escritos específicamente con la plataforma.

En C# tenemos los siguientes ***Tipos básicos o internos***:

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.