

MANUAL DE APRENDIZAJE

PLANTEL: TLACOTEPEC EXT.
TECAMACHALCO

CCT: 21ETC0015V

NOMBRE DE LA ASIGNATURA:
**MI-SII APLICA ESTRUCTURAS DE
CONTROL CON UN LENGUAJE DE
PROGRAMACIÓN**

ELABORADO POR:

Víctor Hugo Ortiz Ramírez

NOMBRE DEL (A) APRENDIENTE:

SEMESTRE Y GRUPO: _____

FECHA DE ELABORACIÓN: **13/02/2023**



SEMESTRE **A**

AGOSTO 2022 – ENERO 2023



Secretaría
de Educación
Gobierno de Puebla

ÍNDICE

BIENVENIDA.....	6
PRESENTACIÓN DE LA ASIGNATURA	6
COMPETENCIAS POR DESARROLLAR.....	7
EXPLICACIÓN DE LA METODOLOGÍA.....	7
PACTO EDUCATIVO.....	8
PRIMER PARCIAL.....	10
ENCUADRE.....	10
PLAN DE EVALUACIÓN.....	11
INICIO.....	12
LO QUE SABES DE UN PROGRAMA.....	12
PARA EMPEZAR CONCEPTOS BÁSICOS DE PROGRAMACIÓN, Y LOS LENGUAJES DE ALTO Y BAJO NIVEL.....	12
RECORDEMOS DIFERENCIA ENTRE LENGUAJES.....	14
LO QUE SABES DE LENGUAJE DE ALTO NIVEL Y BAJO NIVEL.....	15
PARA EMPEZAR ¿QUÉ ES UN COMPILADOR?, ¿Qué ES UN INTERPRETE?	15
RECORDEMOS LA DIFERENCIA ENTRE COMPILADOR E INTERPRETE	17
LO QUE SABES DE PSEUDOCÓDIGO	18
PARA EMPEZAR pseudocódigo.....	18
RECORDEMOS EL USO DE PSEUDOCÓDIGO.....	19
LO QUE SABES DE WRITE Y PRINT	20
PARA EMPEZAR ANÁLISIS DE PRIMERAS LINEAS DE CÓDIGO	20
RECORDEMOS CÓMO FUNCIONAN LAS LINEAS DE UN PROGRAMA.....	21
MANOS A LA OBRA PRIMER PRODUCTO	22
LO QUE SABES DE SOBRE LOS NÚMEROS ENTEROS	27
PARA EMPEZAR, MOSTRAR NÚMEROS ENTEROS EN PANTALLA.....	27
RECORDEMOS LA INTERACCIÓN DE NÚMEROS ENTEROS.....	29
MANOS A LA OBRA PRIMER PRODUCTO	29
LO QUE SABES DE VARIABLES EN LA PROGRAMACIÓN	34
PARA EMPEZAR INTRODUCCIÓN A LAS VARIABLES INT	34
RECORDEMOS como se declaran las variables de tipo int	37
MANOS A LA OBRA TERCER PRODUCTO.....	37
LO QUE SABES SOBRE UN IDENTIFICADOR.....	39
PARA EMPEZAR IDENTIFICADORES.....	39
RECORDEMOS IDENTIFICADORES Y COMENTARIOS.....	42

LO QUE SABES DEL MÉTODO INT 32	42
PARA EMPEZAR A UTILIZAR READLINE	42
RECORDEMOS EL USO DE READLINE	43
LO QUE SABES DE.....	44
PARA EMPEZAR ESTRUCTURA DE CONTROL	45
RECORDEMOS como funciona el if.....	46
MANOS A LA OBRA CON EL CUARTO PRODUCTO	46
SEGUNDO PARCIAL.....	52
ENCUADRE.....	52
PLAN DE EVALUACIÓN.....	53
INICIO	53
LO QUE SABES DE LOS CONDICIONALES.....	53
PARA EMPEZAR ESTRUCTURA CONDICIONAL IF – ELSE	54
RECORDEMOS el uso del if – else.....	57
LO QUE SABES DE ESTRUCTURAS REPETITIVAS	57
PARA EMPEZAR ESTRUCTURA REPETITIVA WHILE - MIENTRAS.....	58
RECORDEMOS el uso de while	60
LO QUE SABES DE BUCLES	60
PARA EMPEZAR ESTRUCTURAS DE CONTROL DO ... WHILE	61
RECORDA EL EMPLEO DE DO WHILE.....	63
MANOS A LA OBRA CON PRIMER PRODUCTO SEGUNDO PARCIAL	63
LO QUE SABES SOBRE LOS TIPOS DE DATOS	70
PARA EMPEZAR TIPOS DE DATOS.....	70
RECORDEMOS USO DE TIPOS DE DATOS.....	74
LO QUE SABES DE CADENA DE CARACTERES	74
PARA EMPEZAR.....	75
RECORDEMOS USO DE LA CADENA DE CARACTERES	76
LO QUE SABES DE DECLARACIÓN DE VARIABLES.....	77
PARA EMPEZAR CON LA DECLARACIÓN DE UNA CLASE Y DEFINICIÓN DE OBJETOS	77
RECORDEMOS EL USO DE DECLARACIÓN DE UNA CLASE Y DEFINICIÓN DE OBJETOS	80
MANOS A LA OBRA CON SEGUNDO PRODUCTO SEGUNDO PARCIAL	80
LO QUE SABES DE UN MÉTODO	86
PARA EMPEZAR DECLARACIÓN DE MÉTODOS.....	86
RECORDEMOS USO DE LOS MÉTODOS	89
MANOS A LA OBRA CON TERCER PRODUCTO SEGUNDO PARCIAL	90

MANOS A LA OBRA	95
CIERRE	96
PRÁCTICA AUTÓNOMA.....	96
EN RESUMEN.....	96
PROYECTO TRANSVERSAL	96
GLOSARIO.....	97
RECURSOS DE APOYO	97
TERCER PARCIAL.....	98
ENCUADRE.....	98
PLAN DE EVALUACIÓN.....	99
INICIO	99
LO QUE SABES DE.....	100
PARA EMPEZAR ciclo for	100
RECORDEMOS USO Y APLIACIÓN DEL CICLO FOR	105
MANOS A LA OBRA CON PRIMER PRODUCTO TERCER PARCIAL	106
LO QUE SABES DE ESTRUCTURA DE SECUENCIAS DE UN PROGRAMA	111
PARA EMPEZAR ESTRUCTURA DE PROGRAMACIÓN SECUENCIAL.....	112
RECORDEMOS USO Y APLIACIÓN DE ESTRUCTURA SECUENCIAL DE PROGRAMACIÓN	114
LO QUE SABES DEL CILO WHILE	114
PARA EMPEZAR CON ESTRUCTURA WHILE.....	114
RECORDEMOS APLICACIÓN Y USO DE CICLO WHILE	123
LO QUE SABES DEL CICLO DO WHILE	123
PARA EMPEZAR EL CICLO DO-WHILE.....	124
RECORDEMOS USO Y APLICACIÓN DE DO-WHILE.....	133
MANOS A LA OBRA CON SEGUNDO PRODUCTO TERCER PARCIAL	134
LO QUE SABES DE ESTRUCTURA DE CONTROL SWITCH	140
PARA EMPEZAR ANALISIS DE LA ESTRUCTURA SWITCH.....	140
Problema 1:	140
Problema 2:	142
RECORDEMOS USO Y APLICACIÓN DE ESTRUCTURA SWITCH	143
MANOS A LA OBRA CON TERCER PRODUCTO TERCER PARCIAL.....	143
GLOSARIO.....	149
RECURSOS DE APOYO	149



Secretaría
de Educación
Gobierno de Puebla



CECYTE
Puebla

COLEGIO DE ESTUDIOS CIENTÍFICOS Y TECNOLÓGICOS
DEL ESTADO DE PUEBLA

Manual de Aprendizaje

BIENVENIDA

Para mis alumnos y alumnas que inician su carrera de programación en el segundo semestre

El estudio requiere de paciencia y flexibilidad mental, pues no todos aprendemos aplicando los mismos métodos. De hecho, en ocasiones, los resultados son cuestión de actitud. No obstante, también sabemos que el estudio puede llegar a ser una actividad agotadora nivel mental y emocional.

Han pasado unas semanas de descanso que cuánta falta y merecido lo tenían, después de trabajar con empeño en todo el semestre escolar pasado. Es un sentimiento inefable el que me embarga de saber que seré su maestro, me siento muy afortunado y feliz... Gracias y espero alcanzar muchos logros con ustedes, estoy seguro de que este semestre será de mucho aprendizaje para todos. Recuerden lo inteligentes, importantes y gentiles que son. Con mucho entusiasmo los invito a ser siempre mejores, cada vez crecen más por lo que, son capaces de hacer más y mejores cosas.

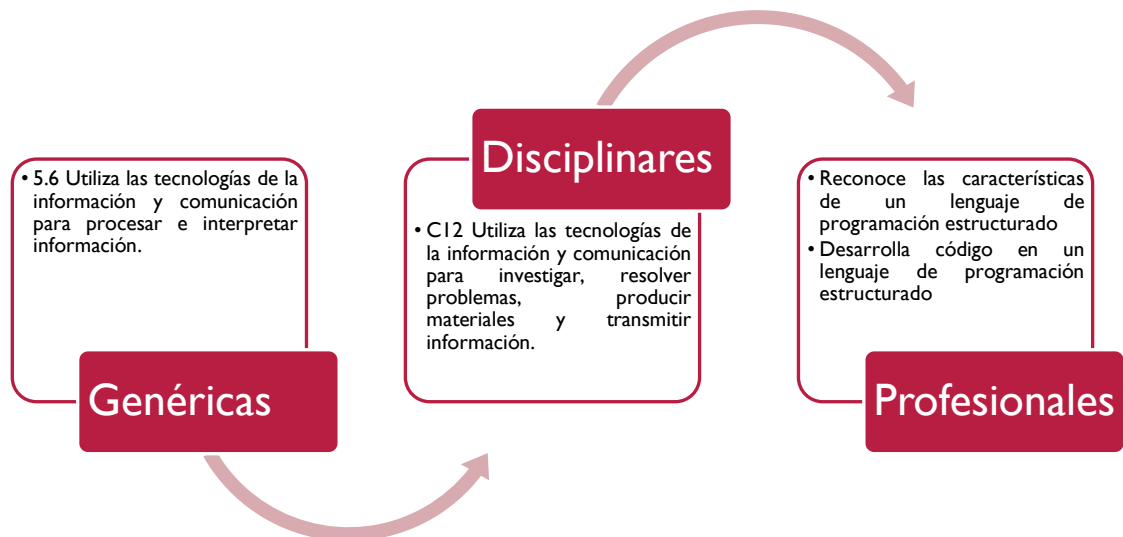
PRESENTACIÓN DE LA ASIGNATURA

Al comenzar a programar, es común que escuchemos el término estructuras de control. Aunque suena como algo aterrador, en este contexto simplemente hace referencia a cómo se ejecuta un programa.

¡Así que no temas! Aquí te presentamos la definición de las estructuras de control en programación, para qué sirven y sus tipos. ¡Comencemos!

C# es un lenguaje de programación desarrollado por Microsoft, orientado a objetos, que ha sido diseñado para compilar diversas aplicaciones que se ejecutan en .NET Framework. Se trata de un lenguaje simple, eficaz y con seguridad de tipos. Las numerosas innovaciones de C# permiten desarrollar aplicaciones rápidamente y mantener la expresividad y elegancia de los lenguajes de estilo de C.

COMPETENCIAS POR DESARROLLAR



EXPLICACIÓN DE LA METODOLOGÍA

HORAS DEL PROGRAMA DE ESTUDIOS	HORAS TEÓRICAS	HORAS PRÁCTICAS
<ul style="list-style-type: none"> •PRIMER PARCIAL •# horas •SEGUNDO PARCIAL •# horas •TERCER PARCIAL •# horas •TOTAL : 	<ul style="list-style-type: none"> •PRIMER PARCIAL •# horas •SEGUNDO PARCIAL •# horas •TERCER PARCIAL •# horas •TOTAL : 	<ul style="list-style-type: none"> •PRIMER PARCIAL •# horas •SEGUNDO PARCIAL •# horas •TERCER PARCIAL •# horas •TOTAL :

PACTO EDUCATIVO

- 1. Asistir puntualmente a clase, con minutos de tolerancia siempre y cuando exista alguna otra actividad o encomienda a los aprendientes.**
- 2. Se hace riguroso el pase de lista en cada clase, tres retardos equivalen a una inasistencia.**
- 3. Si faltó a clase, tengo la obligación de ponerme al corriente y pedir la tarea.**
- 4. Los días lunes, martes y miércoles tengo clase de MI-SII por lo cual no hay pretexto de que no traje mi manual de estudiante y mi USB**
- 5. No se revisan productos, sin portada y sin folder.**
- 6. Queda estrictamente prohibido rayar bancas, sillas, paredes puertas y vidrios.**
- 7. Queda estrictamente prohibido introducir alimentos y refrescos al salón de clases y al laboratorio.**
- 8. Queda prohibido elaborar tareas o trabajos de otras materias durante la clase, persona que sea sorprendida, se le recogerá la tarea y se hará acreedor a la sanción correspondiente.**
- 9. Queda prohibido tirar basura. Al término de la clase, el salón y laboratorio debe quedar limpio y el alumno que haya generado basura tendrá la obligación de recogerla.**
- 10. Queda estrictamente prohibido decir palabras obscenas.**
- 11. Queda estrictamente prohibido el uso de celular, tanto en clases como en exámenes(su uso es solo cuando se indique).**
- 12. Queda estrictamente prohibido poner y decir apodos a los compañeros, así como burlarse porque se equivoquen en clase.**
- 13. Tengo la obligación de poner toda mi atención a la clase, de lo contrario aceptare la sanción que se me asigne(invitación a que el tutor venga a tomar clases con su hijo o hija).**
- 14. Por ningún motivo puedo estar fuera del salón, aunque haya sonado el timbre de termino de clase o no tenga maestro, únicamente con el pase de salida o con el permiso del maestro correspondiente**
- 15. Respetar los criterios de evaluación, así como las fechas de entrega de tareas, productos de aprendizaje.**
- 16. Ante todo respetar al profesor, mis compañeros y este reglamento.**
- 17. Traer las herramientas de trabajo solicitadas por el profesor**
- 18. Evitar traer pertenencias de valor**
- 19. Queda prohibido guardar trabajos en los equipos de cómputo, ya que usualmente se borran para liberar almacenamiento.**

ENTERADOS:



Firma de conformidad alumno

**Firma de conformidad tutor
padre**



PRIMER PARCIAL

ENCUADRE

COMPETENCIA PROFESIONAL

Reconoce las características de un lenguaje de programación estructurado

Desarrolla código en un lenguaje de programación estructurado

SITUACIÓN DE APRENDIZAJE



PLAN DE EVALUACIÓN

PRODUCTOS DE APRENDIZAJE PRIMER PARCIAL			
PRODUCTO	PONDERACIÓN	PORCENTAJE OBTENIDO	OBSERVACIONES
Primer programa	15%		
Compendio de programas	20%		
Analogías para programas	15%		
Confección de programa if	30%		
Tutorías y Orientación Educativa	10%		
Innovación y liderazgo CECyTE	10%		

INICIO

Aprendizajes claves

LO QUE SABES DE UN PROGRAMA



Un programa es un conjunto de órdenes para un ordenador.

PARA EMPEZAR CONCEPTOS BÁSICOS DE PROGRAMACIÓN, Y LOS LENGUAJES DE ALTO Y BAJO NIVEL

Conceptos básicos sobre programación

Un programa es un conjunto de órdenes para un ordenador. Estas órdenes se le deben dar en un cierto lenguaje, que el ordenador sea capaz de comprender.

El problema es que los lenguajes que realmente entienden los ordenadores resultan difíciles para nosotros, porque son muy distintos de los que nosotros empleamos habitualmente para hablar. Escribir programas en el lenguaje que utiliza internamente el ordenador (llamado "lenguaje máquina" o "código máquina") es un trabajo duro, tanto a la hora de crear el programa como (especialmente) en el momento de corregir algún fallo o mejorar lo que se hizo.

Por ejemplo, un programa que simplemente guardara un valor "2" en la posición de memoria 1 de un ordenador sencillo, con una arquitectura propia de los años 80, basada en el procesador Z80 de 8 bits, sería así en código máquina:

```
0011 1110 0000 0010 0011 1010 0001 0000
```

Prácticamente ilegible. Por eso, en la práctica se emplean lenguajes más parecidos al lenguaje humano, llamados "lenguajes de alto nivel". Normalmente, estos son muy parecidos al idioma inglés, aunque siguen unas reglas mucho más estrictas.

Lenguajes de alto nivel y de bajo nivel.

Vamos a ver en primer lugar algún ejemplo de lenguaje de alto nivel, para después comparar con lenguajes de bajo nivel, que son los más cercanos al ordenador.

Uno de los lenguajes de alto nivel más sencillos es el lenguaje BASIC. En este lenguaje, escribir el texto Hola en pantalla, sería tan sencillo como usar la orden

```
program Saludo;  
  
begin  
  write('Hola');  
end.
```

El equivalente en lenguaje C resulta algo más difícil de leer:

```
#include <stdio.h>  
  
int main()  
{  
  printf("Hola");  
  
}
```

En C# hay que dar todavía más pasos para conseguir lo mismo:

```
public class Ejemplo01  
{  
  public static void Main()  
  {  
    System.Console.WriteLine("Hola");  
  }  
}
```

Como se puede observar, a medida que los lenguajes evolucionan, son capaces de ayudar al programador en más tareas, pero a la vez, los programas sencillos se vuelven más complicados. Afortunadamente, no todos los lenguajes siguen esta regla, y algunos se han diseñado de forma que las tareas sencillas sean sencillas de programar (de nuevo). Por ejemplo, para escribir algo en pantalla usando el lenguaje Python haríamos:

```
print("Hello")
```

Por el contrario, los lenguajes de bajo nivel son más cercanos al ordenador que a los lenguajes humanos. Eso hace que sean más difíciles de aprender y también que los fallos sean más difíciles de descubrir y corregir, a cambio de que podemos optimizar

al máximo la velocidad (si sabemos cómo), e incluso llegar a un nivel de control del ordenador que a veces no se puede alcanzar con otros lenguajes. Por ejemplo, escribir Hola en lenguaje ensamblador de un ordenador equipado con el sistema operativo MsDos y con un procesador de la familia Intel x86 sería algo como

```
dosseg
.model small
.stack 100h

.data
hello_message db 'Hola',0dh,0ah,'$'

.code
main proc
    mov     ax,@data
    mov     ds,ax

    mov     ah,9
    mov     dx,offset hello_message
    int     21h

    mov     ax,4C00h
    int     21h
main endp
end main
```

Resulta bastante más difícil de seguir. Pero eso todavía no es lo que el ordenador entiende, aunque tiene una equivalencia casi directa. Lo que el ordenador realmente es capaz de comprender son secuencias de ceros y unos. Por ejemplo, las órdenes "mov ds, ax" y "mov ah, 9" (en cuyo significado no vamos a entrar) se convertirían a

RECORDEMOS DIFERENCIA ENTRE LENGUAJES

Lenguaje de alto nivel	Lenguaje de bajo nivel
<ul style="list-style-type: none">• Lenguaje sencillo lenguaje BASIC• Ocupa un lenguaje sencillo y fácil de entender como el uso de inglés técnico.•	<ul style="list-style-type: none">• los lenguajes de bajo nivel son más cercanos al ordenador que a los lenguajes humanos.• Lo que el ordenador realmente es capaz de comprender son secuencias de ceros y unos

LO QUE SABES DE LENGUAJE DE ALTO NIVEL Y BAJO NIVEL



Compilar: Es un Software que traduce un programa escrito en un lenguaje de programación de alto nivel (C / C ++, COBOL, etc.)

Interpretes: Los intérpretes son programas que, a diferencia de un compilador, no leen todo el código primero como un todo

PARA EMPEZAR ¿QUÉ ES UN COMPILADOR?, ¿QUÉ ES UN INTERPRETE?

¿Qué es un compilador?

Un compilador es uno de los pilares de la programación y de cómo entender la comunicación entre un lenguaje de alto nivel y una máquina. Al poder conocer el funcionamiento de este paso intermedio nos permitirá desarrollar y programar de una forma más precisa los lenguajes de alto nivel.

Tradicionalmente los compiladores generaban código máquina de inferior calidad que el que podían escribir programadores humanos, pero actualmente los compiladores proporcionan hoy en día un código máquina de alta calidad pequeño y rápido, haciendo poco atractiva la programación en ensamblador, programación que en asignaturas como está ya simplemente se menciona por conocerla pero no se realiza un estudio para aprender este tipo de programación.

Los programadores de ensamblador siguen teniendo ventaja en cuanto a que disponen de un mayor conocimiento global del programa que les permite realizar determinadas optimizaciones del código que resultan muy difíciles para los compiladores.

¿En qué se diferencia de un Intérprete?

Para responder a esta pregunta primero debemos conocer que es y para qué sirve un Intérprete.

Un intérprete lee un programa fuente ejecutable, escrito en un lenguaje de programación de alto nivel, así como datos para este programa, y ejecuta el programa contra los datos para producir algunos resultados. Un ejemplo es el intérprete de shell de Unix, que ejecuta comandos del sistema operativo de forma interactiva.

Hay que tener en cuenta que tanto los intérpretes como los compiladores (como cualquier otro programa) están escritos en un lenguaje de programación de alto nivel (que puede ser diferente del idioma que aceptan) y se traducen en código máquina.

Por ejemplo, un intérprete de Java puede escribirse completamente en C o incluso en Java. El programa fuente del intérprete es independiente de la máquina ya que no genera código de máquina.

Un intérprete generalmente es más lento que un compilador porque procesa e interpreta cada enunciado de un programa tantas veces como el número de evaluaciones de esta afirmación. Por ejemplo, cuando se interpreta un bucle for, las afirmaciones dentro del cuerpo for-loop se analizarán y evaluarán en cada paso del bucle. Algunos lenguajes, como Java y Lisp, vienen con un intérprete y un compilador. Los programas fuente de Java (clases Java con extensión .java) son traducidos por el compilador javac en archivos de códigos de bytes (con extensión .class).

El intérprete de Java, llamado Java Virtual Machine (JVM), en realidad puede interpretar códigos de bytes directamente o puede compilarlos internamente en código máquina y luego ejecutar ese código.

Los compiladores son procesos complejos debido a que tienen varias fases por las que un programa fuente debe de pasar antes de convertirse en un programa ejecutable, los pasos son los siguiente

Analizador léxico:

El analizador léxico o lexicográfico (Scanner en inglés) es la primera etapa del proceso de compilación, el cual se encarga de dividir el programa en Tokens, los cuales, según una tabla de símbolos definida por el mismo lenguaje.

De esta forma cada token del programa es clasificado según su significado para ser procesados en la segunda etapa del proceso de compilación.

Analizador sintáctico:

El analizador sintáctico (Parse en inglés), es la segunda fase del proceso de compilación y tiene como finalidad la generación de un Árbol sintáctico, el cual no es más que una estructura de datos compleja que permite representar de una forma más simple al programa fuente.

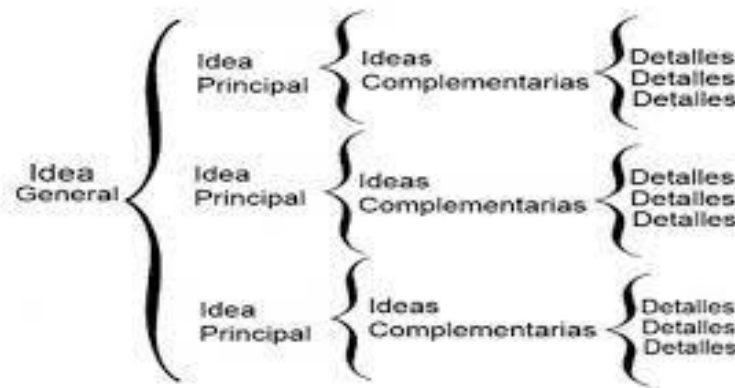
Los compiladores modernos utilizan estructuras de objetos para representa a un programa, de esta forma existe una clase específica para representa cada posible token de nuestra tabla de símbolos.

Analizador semántico:

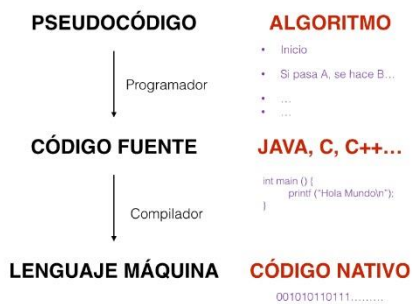
El analizador semántico es el último paso antes de empezar a compilar realmente el código, prepara el programa para ser compilado. El analizador semántico parte del árbol sintáctico abstracto y tiene la finalidad de validar los puntos más finos del programa, como por ejemplo, validar compatibilidad en tipos de datos, que la variable utilizada en una instrucción este previamente declara o que estén dentro del contexto, si implementamos una interface que todos los métodos estén definidos, etc.

RECORDEMOS LA DIFERENCIA ENTRE COMPILADOR E INTERPRETE

Realizar un cuadro sinóptico del tema visto, extrayendo las ideas principales, con la finalidad de que refuerces tus conocimientos.



LO QUE SABES DE PSEUDOCÓDIGO



El pseudocódigo es una forma de expresar los distintos pasos que va a realizar un programa, de la forma más parecida a un lenguaje de programación.

PARA EMPEZAR PSEUDOCÓDIGO

Pseudocódigo A pesar de que los lenguajes de alto nivel se acercan al lenguaje natural, que nosotros empleamos, es habitual no usar ningún lenguaje de programación concreto cuando queremos plantear los pasos necesarios para resolver un problema, sino emplear un lenguaje de programación ficticio, no tan estricto, muchas veces escrito incluso en español. Este lenguaje recibe el nombre de **pseudocódigo**.

Esa secuencia de pasos para resolver un problema es lo que se conoce como **algoritmo** (realmente hay alguna condición más, por ejemplo, debe ser un número finito de pasos). Por tanto, un programa de ordenador es un algoritmo expresado en un lenguaje de programación.

Por ejemplo, un algoritmo que controlase los pagos que se realizan en una tienda con tarjeta de crédito, escrito en pseudocódigo, podría ser:

```
Leer banda magnética de la tarjeta
Conectar con central de cobros
Si hay conexión y la tarjeta es correcta:
  Pedir código PIN
  Si el PIN es correcto
    Comprobar saldo_existente
    Si saldo_existente >= importe_compra
      Aceptar la venta
      Descontar importe del saldo.
    Fin Si
  Fin Si
Fin Si
```

Dadas dos variables numéricas A y B, que el usuario debe teclear, se pide realizar un algoritmo que intercambie los valores de ambas variables y muestre cuanto valen al final las dos variables (recuerda la asignación).

```
1  Proceso ejercicio_1
2    Escribir "Introduce el valor de A"
3    Leer A
4    Escribir "Introduce el valor de B"
5    Leer B
6    C<-A
7    A<-B
8    B<-C
9    Escribir "A vale " A " y B vale " B
10 FinProceso
```

RECORDEMOS EL USO DE PSEUDOCÓDIGO

1. Ejercicios propuestos

1.1 Algoritmo que lea dos números, calculando y escribiendo el valor de su suma, resta, producto y división.

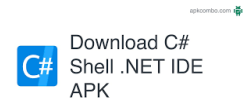
1.2 Algoritmo que lea dos números y nos diga cual de ellos es mayor o bien si son iguales (recuerda usar la estructura condicional SI).

2. Ejercicio

2.1. Localizar en Internet Visual Studio y realizar la instalación en tu equipo de cómputo el cual tiene más herramientas y beneficios de utilizarlo en el escritorio y familiarizarte con la herramienta o de lo contrario seguir el paso 2.



2.2. Localizar en Google Play, app aplicación de **C# Shell.Net IDE**, verificar la instalación, además de familiarizarte con la herramienta



LO QUE SABES DE WRITE Y PRINT



Escribir: write, **Imprimir:** print

PARA EMPEZAR ANÁLISIS DE PRIMERAS LINEAS DE CÓDIGO

Análisis de las primeras líneas de código

Escribir un texto en C# Vamos con un primer ejemplo de programa en C#, posiblemente el más sencillo de los que "hacen algo útil". Se trata de escribir un texto en pantalla. La apariencia de este programa la vimos en el tema anterior. Vamos a verlo ahora con más detalle:

```
public class Ejemplo01
{
    public static void Main()
    {
        System.Console.WriteLine("Hola");
    }
}
```

Esto escribe "Hola" en la pantalla. Pero hay mucho alrededor de ese "Hola", y vamos a comentarlo antes de proseguir, aunque muchos de los detalles se irán aclarando más adelante. En este primer análisis, iremos de dentro hacia fuera:

Análisis del código

- `WriteLine("Hola");` - "Hola" es el texto que queremos escribir, y `WriteLine` es la orden encargada de escribir (`Write`) una línea (`Line`) de texto en pantalla.
- `Console.WriteLine("Hola");` porque `WriteLine` es una orden de manejo de la "consola" (la pantalla "negra" en modo texto del sistema operativo).
- `System.Console.WriteLine("Hola");` porque las órdenes relacionadas con el manejo de consola (`Console`) pertenecen a la categoría de sistema (`System`).
- Las llaves `{ y }` se usan para delimitar un bloque de programa. En nuestro caso, se trata del bloque principal del programa.
- **Public static void** `Main()` - `Main` indica cual es "el cuerpo del programa", la parte principal (un programa puede estar dividido en varios fragmentos, como veremos más adelante). Todos los programas tienen que tener un bloque "Main". Los detalles de por qué hay que poner delante "public static void" y de por qué se pone después un paréntesis vacío los iremos aclarando más tarde. De momento, deberemos memorizar que ésa será la forma correcta de escribir "Main".
- **Public class** `Ejemplo01` - de momento pensaremos que "Ejemplo01" es el nombre de nuestro programa. Una línea como esa deberá existir también siempre en nuestros programas, y eso de "public class" será obligatorio. Nuevamente, aplazamos para más tarde los detalles sobre qué quiere decir "class" y por qué debe ser "public".

Como se puede ver, mucha parte de este programa todavía es casi un "acto de fe" para nosotros. Debemos creernos que "se debe hacer así". Poco a poco iremos detallando el por qué de "public", de "static", de "void", de "class"... Por ahora nos limitaremos a "rellenar" el cuerpo del programa para entender los conceptos básicos de programación.

RECORDEMOS CÓMO FUNCIONAN LAS LINEAS DE UN PROGRAMA

Crea un programa en C# que te salude por tu nombre (ej: "Hola, Nacho").

Sólo un par de cosas más antes de seguir adelante:

- Cada orden de C# debe terminar con un punto y coma (;)
- C# es un lenguaje de formato libre, de modo que puede haber varias órdenes en una misma línea, u órdenes separadas por varias líneas o espacios entre medias. Lo que realmente indica donde termina una orden y donde empieza la siguiente son los puntos y coma. Por ese motivo, el programa anterior se podría haber escrito también así (aunque no es aconsejable, porque puede resultar menos legible)

MOMENTO DE ACTIVIDAD I

MANOS A LA OBRA PRIMER PRODUCTO

Nombre de la práctica: Hola mundo

Nomenclatura: MI-SII Primer Programa/Hola mundo

Tema: Primeras líneas de código

Duración: 5 Sesiones

Objetivos: *(general y específico) Utilizar la herramientas de programación, mediante el uso de las primeras líneas de código en lenguaje de C#*

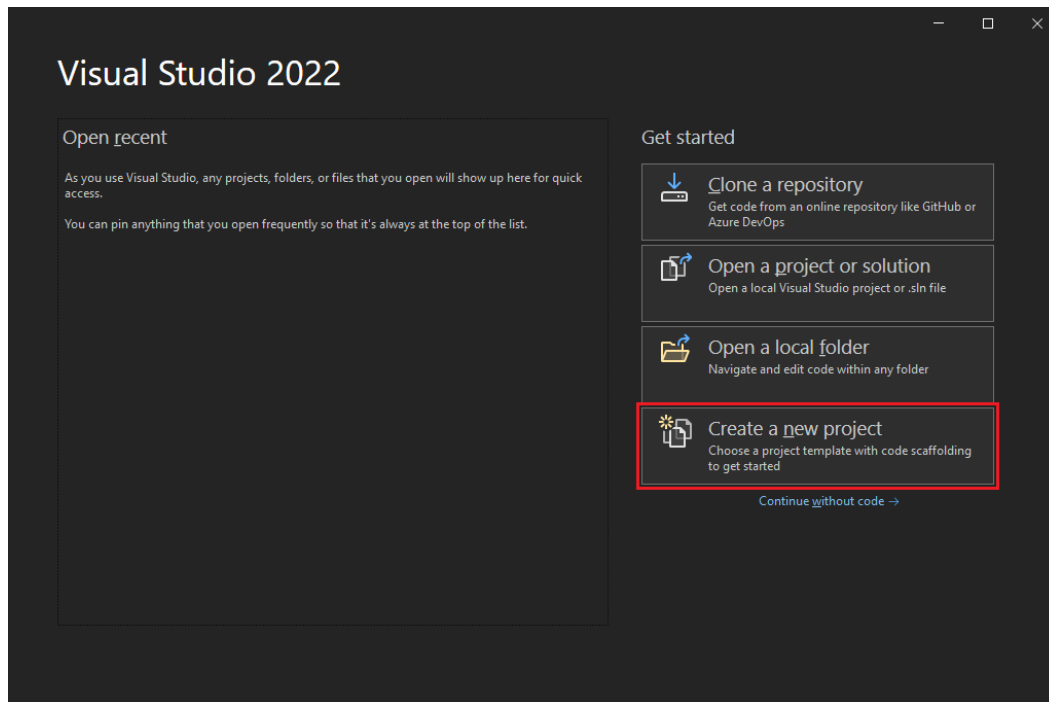
Materiales y equipo: Equipo de computo con software instalado Visual Studio o en su caso aplicación de Play Store

Procedimiento:

Crear un proyecto

Para empezar, cree un proyecto de aplicación de C#. En el tipo de proyecto se incluyen todos los archivos de plantilla que necesita.

I. Abra Visual Studio y elija **Crear un nuevo proyecto** en la ventana Inicio.

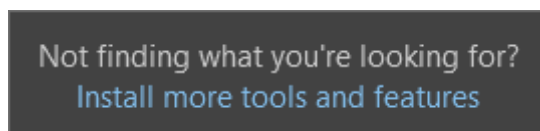


2. En la ventana **Crear un nuevo proyecto**, seleccione **Todos los lenguajes** y, a continuación, **elija C#** en la lista desplegable. Seleccione **Windows** en la lista **Todas las plataformas** y **Consola** en la lista **Todos los tipos de proyecto**.

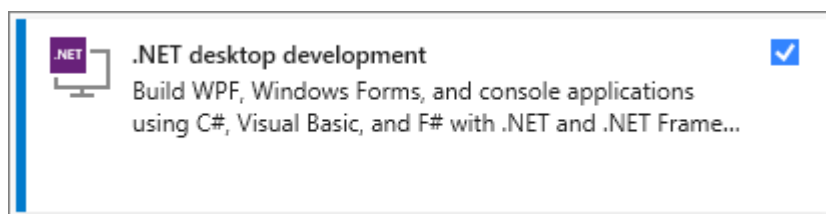
Después de aplicar los filtros de lenguaje, plataforma y tipo de proyecto, elija la plantilla **Aplicación de consola** y, luego, seleccione **Siguiente**.

Nota

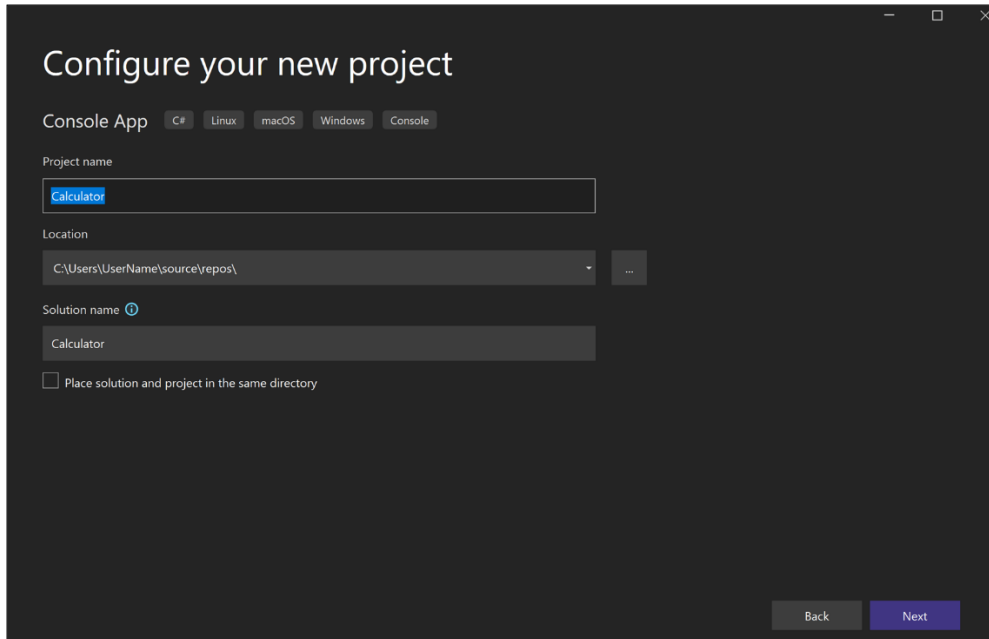
Si no ve la plantilla **Aplicación de consola**, seleccione **Instalar más herramientas y características**.



En el Instalador de Visual Studio, elija la carga de trabajo de **desarrollo de escritorio de .NET** y, a continuación, seleccione **Modificar**.



3. En la ventana **Configurar el nuevo proyecto**, escriba CALCULATOR en el cuadro **Nombre del proyecto** y, a continuación, seleccione **Siguiente**.



Configure your new project

Console App C# Linux macOS Windows Console

Project name

Calculator

Location

C:\Users\UserName\source\repos\

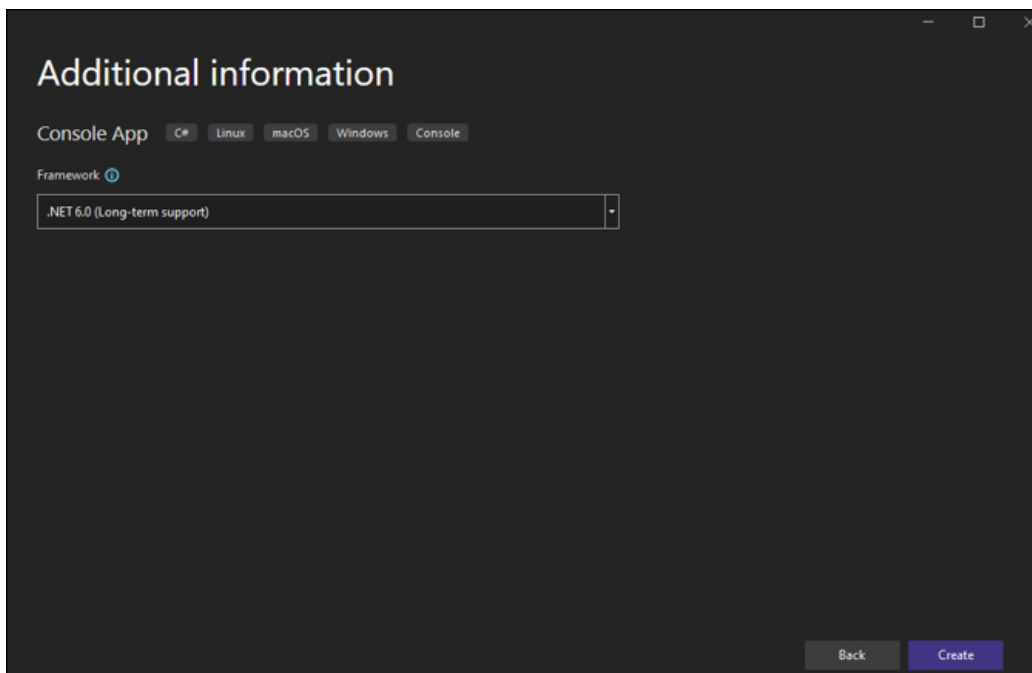
Solution name ⓘ

Calculator

☐ Place solution and project in the same directory

Back Next

4. En la ventana **Información adicional**, **.NET 6.0** ya debe estar seleccionado para la plataforma de destino. Seleccione **Crear**.



Additional information

Console App C# Linux macOS Windows Console

Framework ⓘ

.NET 6.0 (Long-term support)

Back Create

Visual Studio abre el nuevo proyecto, que incluye código predeterminado de "Hola mundo".

Para verlo en el editor, seleccione el archivo de código Program.cs en la ventana del Explorador de soluciones, que normalmente se encuentra en el lado derecho de Visual Studio.

La instrucción de código única llama al método WriteLine para mostrar la cadena literal "¡Hola mundo!" en la ventana de consola. Si presiona F5, puede ejecutar el programa predeterminado en modo de depuración. Una vez que la aplicación se ejecuta en el depurador, la ventana de la consola permanece abierta. Presione cualquier tecla para cerrar la ventana de consola.

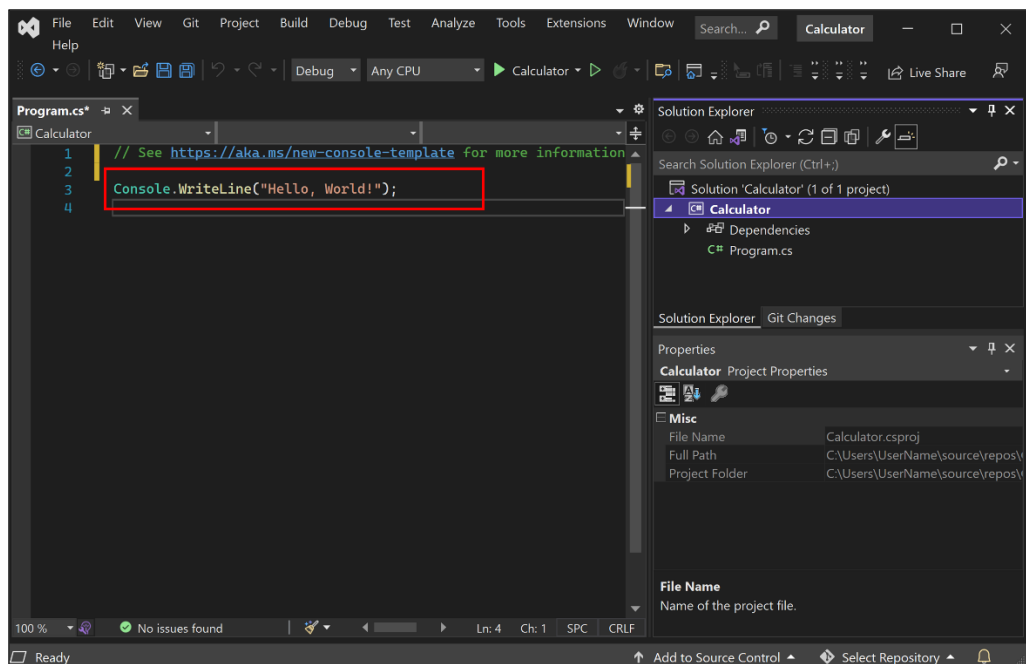
Nota

A partir de .NET 6, los nuevos proyectos que usan la plantilla de consola generan código diferente al de versiones anteriores. Para obtener más información, consulte la página Las nuevas plantillas de C# generan instrucciones de nivel superior.

Creación de la aplicación

Empiece con algunos cálculos básicos de enteros en C#.

1. En el **Explorador de soluciones**, en el panel derecho, seleccione **Program.cs** para mostrar el archivo en el editor de código
2. En el editor de código, reemplace el código predeterminado "Hello World" que dice `Console.WriteLine("Hello, World!");`.



Reemplace la línea por el código siguiente, por el código personal de saludo



Hola Pedrito

INSTRUMENTO DE EVALUACIÓN

Aspectos a evaluar	Ponderación		
	Porcentaje	sí	No
Portada con los datos de identificación	2		
La impresión de pantalla es clara y facil de interpretar	3		
Contiene el codigo visible y de fácil interpretación	3		
el mensaje tiene los datos de saludo, haciendo referencia al aprendiente	3		
Tiene limpieza y formalidad	2		
Se entrega con un folder en condiciones optimas de un trabajo de media superior	2		
Nombre y firma del docente	Nombre y firma del padre tutor		

A LO QUE LLEGAMOS

Menciona la importancia de la programación en tu vida cotidiana:

LO QUE SABES DE SOBRE LOS NÚMEROS ENTEROS



Los números enteros pueden sumarse, restarse, multiplicarse o dividirse tal y como los números naturales, pero siempre obedeciendo a las normas que determinan el signo resultante, de la siguiente manera:

PARA EMPEZAR, MOSTRAR NÚMEROS ENTEROS EN PANTALLA

Mostrar números enteros en pantalla Cuando queremos escribir un texto "tal cual", como en el ejemplo anterior, lo encerramos entre comillas. Pero no siempre queremos escribir textos prefijados. En muchos casos, se tratará de algo que habrá que calcular.

El ejemplo más sencillo es el de una operación matemática. La forma de realizarla es sencilla: no usar comillas en WriteLine. Entonces, C# intentará analizar el contenido para ver qué quiere decir. Por ejemplo, para sumar 3 y 4 bastaría hacer:

```
public class Ejemplo01suma
{
    public static void Main()
    {
        System.Console.WriteLine(3+4);
    }
}
```

Operaciones aritméticas

Operadores

Está claro que el símbolo de la suma será un +, y podemos esperar cual será el de la resta, pero alguna de las operaciones matemáticas habituales tiene símbolos menos intuitivos. Veamos cuales son los más importantes:

Operador	Operación
+	Suma
-	Resta, negación
*	Multiplicación
/	División
%	Resto de la división ("módulo")

Orden de prioridad de los operadores

- ☐ En primer lugar se realizarán las operaciones indicadas entre paréntesis.
- ☐ Luego la negación.

- ☐ Después las multiplicaciones, divisiones y el resto de la división.
- ☐ Finalmente, las sumas y las restas.
- ☐ En caso de tener igual prioridad, se analizan de izquierda a derecha.

RECORDEMOS LA INTERACCIÓN DE NÚMEROS ENTEROS

- Crea un programa que diga el resultado de sumar 118 y 56.
- Crea un programa que diga el resultado de sumar 12345 y 67890.
- Hacer un programa que calcule el producto de los números 12 y 13.
- Hacer un programa que calcule la diferencia (resta) entre 321 y 213.
- Hacer un programa que calcule el resultado de dividir 301 entre 3.

MANOS A LA OBRA PRIMER PRODUCTO

Nombre de la práctica: Compendio de programas

Nomenclatura: MI-SII Practica I/compendio de programas

Tema: Primeras líneas de código

Duración: 5 Sesiones

Objetivos: *Utilizar las herramientas de programación, mediante el uso de las primeras líneas de código en lenguaje y empleo de operaciones básicas en la programación*

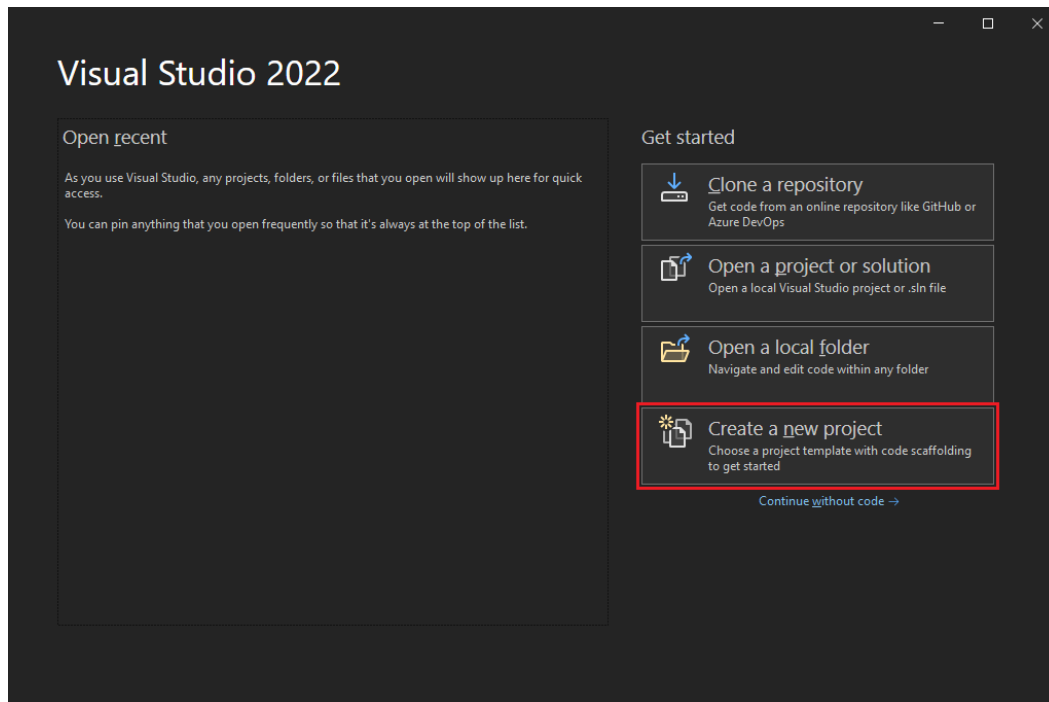
Materiales y equipo: Equipo de cómputo con software instalado Visual Studio o en su caso aplicación de Play Store

Procedimiento:

Crear un proyecto

- I. Para empezar, cree un proyecto de aplicación de C#. En el tipo de proyecto se incluyen todos los archivos de plantilla que necesita.

Abra Visual Studio y elija **Crear un nuevo proyecto** en la ventana Inicio.

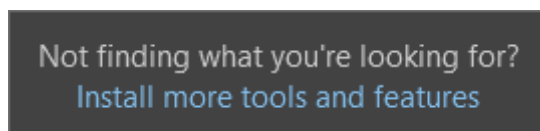


2. En la ventana **Crear un nuevo proyecto**, seleccione **Todos los lenguajes** y, a continuación, elija **C#** en la lista desplegable. Seleccione **Windows** en la lista **Todas las plataformas** y **Consola** en la lista **Todos los tipos de proyecto**.

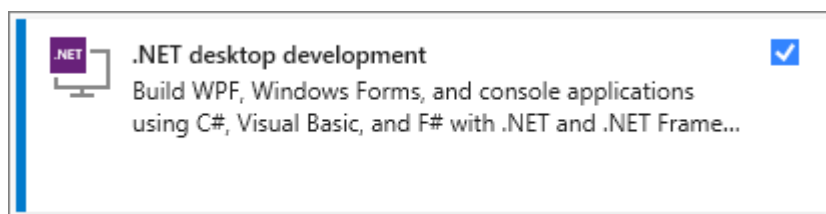
Después de aplicar los filtros de lenguaje, plataforma y tipo de proyecto, elija la plantilla **Aplicación de consola** y, luego, seleccione **Siguiente**.

Nota

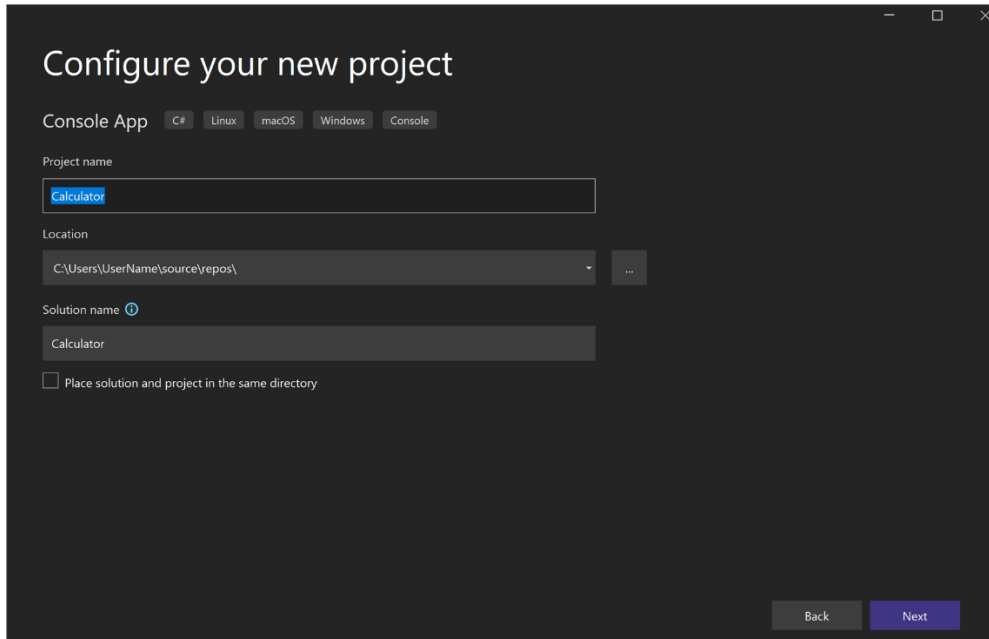
Si no ve la plantilla **Aplicación de consola**, seleccione **Instalar más herramientas y características**.



En el Instalador de Visual Studio, elija la carga de trabajo de **desarrollo de escritorio de .NET** y, a continuación, seleccione **Modificar**.



3. En la ventana **Configurar el nuevo proyecto**, escriba CALCULATOR en el cuadro **Nombre del proyecto** y, a continuación, seleccione **Siguiente**.



Configure your new project

Console App C# Linux macOS Windows Console

Project name

Calculator

Location

C:\Users\UserName\source\repos\

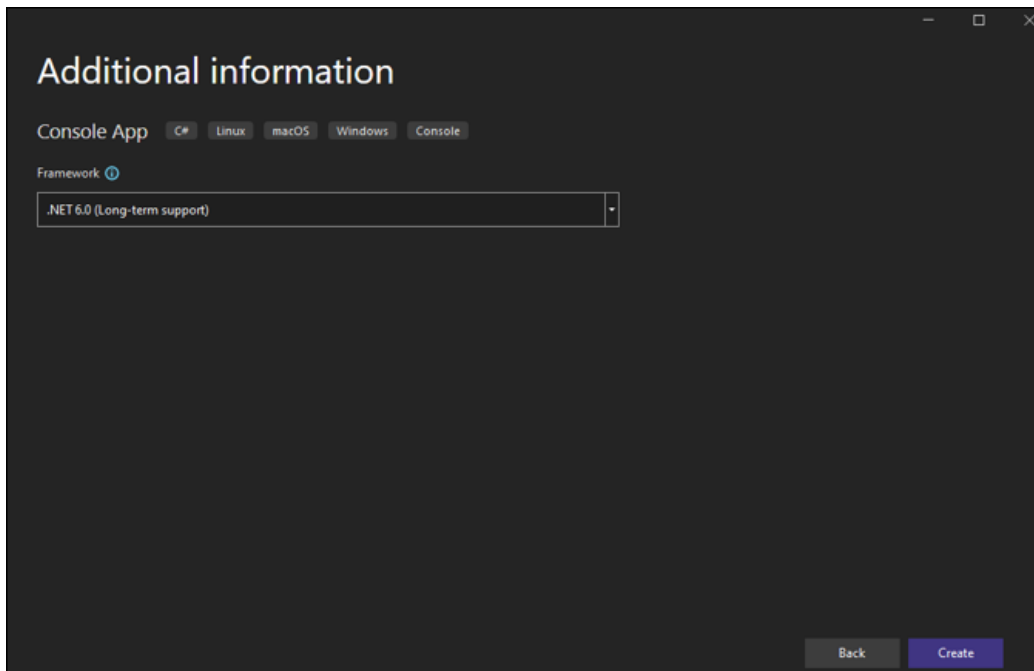
Solution name ⓘ

Calculator

☐ Place solution and project in the same directory

Back Next

4. En la ventana **Información adicional**, **.NET 6.0** ya debe estar seleccionado para la plataforma de destino. Seleccione **Crear**.



Additional information

Console App C# Linux macOS Windows Console

Framework ⓘ

.NET 6.0 (Long-term support)

Back Create

Visual Studio abre el nuevo proyecto, que incluye código predeterminado de "Hola mundo".

Para verlo en el editor, seleccione el archivo de código Program.cs en la ventana del Explorador de soluciones, que normalmente se encuentra en el lado derecho de Visual Studio.

La instrucción de código única llama al método WriteLine para mostrar la cadena literal "¡Hola mundo!" en la ventana de consola. Si presiona F5, puede ejecutar el programa predeterminado en modo de depuración. Una vez que la aplicación se ejecuta en el depurador, la ventana de la consola permanece abierta. Presione cualquier tecla para cerrar la ventana de consola.

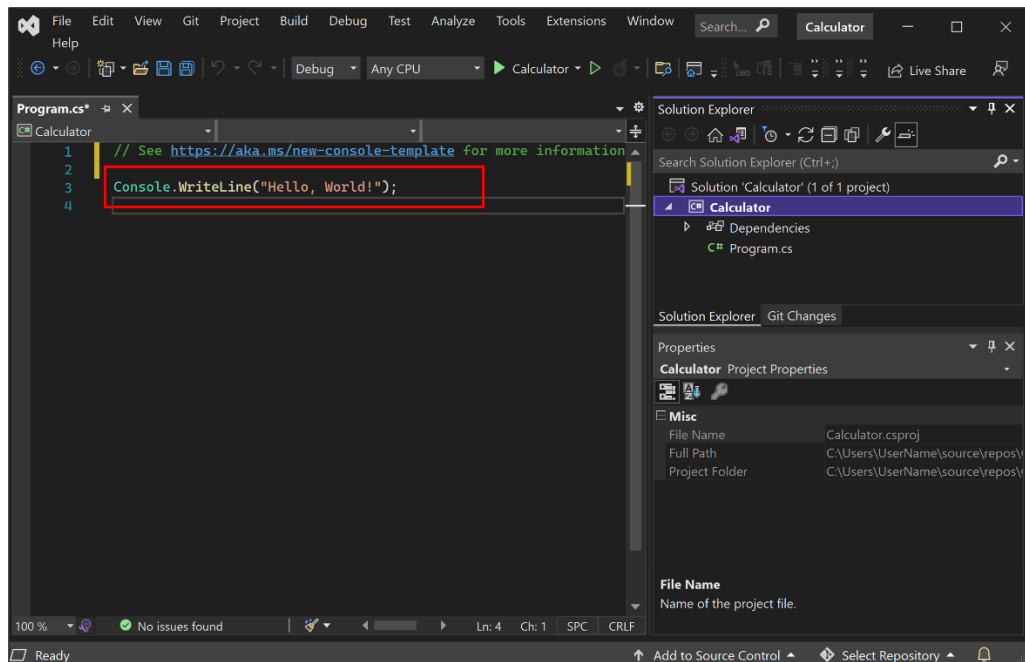
Nota

A partir de .NET 6, los nuevos proyectos que usan la plantilla de consola generan código diferente al de versiones anteriores. Para obtener más información, consulte la página Las nuevas plantillas de C# generan instrucciones de nivel superior.

Creación de la aplicación

Empiece con algunos cálculos básicos de enteros en C#.

5. En el **Explorador de soluciones**, en el panel derecho, seleccione **Program.cs** para mostrar el archivo en el editor de código
6. En el editor de código, reemplace el código predeterminado "Hello World" que dice `Console.WriteLine("Hello World!");`.



7. Codifica cada uno de los enunciados del programa, siguiendo los procesos uno a uno:

- Crea un programa que diga el resultado de sumar 118 y 56.
- Crea un programa que diga el resultado de sumar 12345 y 67890.
- Hacer un programa que calcule el producto de los números 12 y 13.
- Hacer un programa que calcule la diferencia (resta) entre 321 y 213.
- Hacer un programa que calcule el resultado de dividir 301 entre 3.

INSTRUMENTO DE EVALUACIÓN DEL SEGUNDO PRODUCTO

Aspectos a evaluar	Ponderación		
	Porcentaje	sí	No
Portada con los datos de identificación	2		
Código del programa de suma de 118 y 56	1.5		
Resultado del programa suma de 118 y 56	1.5		
Código del programa de sumar 12345 y 67890	1.5		
Resultado del programa de sumar 12345 y 67890	1.5		
Código del programa del producto de los números 12 y 13	1.5		
Resultado del programa del producto de los números 12 y 13	1.5		
Código del programa de calcule la diferencia (resta) entre 321 y 213	1.5		
Resultado del programa de calcule la diferencia (resta) entre 321 y 213	1.5		
Código del programa que calcule el resultado de dividir 301 entre 3.	1.5		
Resultado del programa que calcule el resultado de dividir 301 entre 3.	1.5		
Tiene limpieza y formalidad	2		
Se entrega con un folder en condiciones óptimas de un trabajo de media superior	1		
Nombre y firma del docente	Nombre y firma del padre tutor		

A LO QUE LLEGAMOS

Plantear interrogaciones que den muestra de lo aprendido en la práctica (reporte de práctica)

LO QUE SABES DE VARIABLES EN LA PROGRAMACIÓN



id	valor
edad	22
apellido	"Chang"
buscando	true
peso	9.35

El concepto de variable se utiliza en muchas disciplinas, el ejemplo más común quizás es la variable x en matemática, que se trata de un valor desconocido que debemos encontrar. En programación las variables son un concepto algo distinto.

Para entender qué es una variable en programación vamos a ver un poco sobre memorias de computadoras.

Las memorias almacenan información en forma de registros de bits (es decir conjuntos de valores que pueden ser 0 o 1). Esta información será interpretada y utilizada por nuestros programas. Las memorias a su vez cuentan con muchos de estos registros, así que cada registro tiene una dirección que permite encontrarlo, leerlo y escribirlo.

Entonces podemos decir que una variable es un DATO que se aloja en la memoria de la computadora. El nombre de identificación que le asignamos está asociado a la dirección dentro de la memoria y el valor que guardamos es la información que contiene la variable.

PARA EMPEZAR INTRODUCCIÓN A LAS VARIABLES INT

Variables int

Las variables son algo que no contiene un valor predeterminado, un espacio de memoria al que nosotros asignamos un nombre y en el que podremos almacenar datos.

El primer ejemplo nos permitía escribir "Hola". El segundo nos permitía sumar dos números que habíamos prefijado en nuestro programa. Pero esto tampoco es "lo habitual", sino que esos números dependerán de valores que haya tecleado el usuario o de cálculos anteriores.

Por eso necesitaremos usar variables, zonas de memoria en las que guardemos los datos con los que vamos a trabajar y también los resultados temporales. Como primer ejemplo, vamos a ver lo que haríamos para sumar dos números enteros que fijásemos en el programa.

Definición de variables: números enteros

Para usar una cierta variable primero hay que **declararla**: indicar su nombre y el tipo de datos que queremos guardar.

El primer tipo de datos que usaremos serán números enteros (sin decimales), que se indican con "int" (abreviatura del inglés "integer"). Después de esta palabra se indica el nombre que tendrá la variable:

```
int primerNumero;
```

Esa orden reserva espacio para almacenar un número entero, que podrá tomar distintos valores, y al que nos referiremos con el nombre "primerNumero".

Asignación de valores

Podemos darle un valor a esa variable durante el programa haciendo

```
primerNumero = 234;
```

O también podemos darles un valor inicial ("inicializarlas") antes de que empiece el programa, en el mismo momento en que las definimos:

```
int primerNumero = 234;
```

O incluso podemos definir e inicializar más de una variable a la vez

```
int primerNumero = 234, segundoNumero = 567;
```

(esta línea reserva espacio para dos variables, que usaremos para almacenar números enteros; una de ellas se llama primerNumero y tiene como valor inicial 234 y la otra se llama segundoNumero y tiene como valor inicial 567).

Después ya podemos hacer operaciones con las variables, igual que las hacíamos con los números:

```
suma = primerNumero + segundoNumero;
```

Mostrar el valor de una variable en pantalla

Una vez que sabemos cómo mostrar un número en pantalla, es sencillo mostrar el valor de una variable. Para un número hacíamos cosas como

```
System.Console.WriteLine(3+4);
```

pero si se trata de una variable es idéntico:

```
System.Console.WriteLine(suma);
```

O bien, si queremos mostrar un texto además del valor de la variable, podemos indicar el texto entre comillas, detallando con {0} en qué parte del texto queremos que aparezca el valor de la variable, de la siguiente forma:

```
System.Console.WriteLine("La suma es {0}", suma);
```

Si se trata de más de una variable, indicaremos todas ellas tras el texto, y detallaremos dónde debe aparecer cada una de ellas, usando {0}, {1} y así sucesivamente:

```
System.Console.WriteLine("La suma de {0} y {1} es {2}",  
    primerNumero, segundoNumero, suma);
```

Ya sabemos todo lo suficiente para crear nuestro programa que sume dos números usando variables:

```
System.Console.WriteLine("La suma de {0} y {1} es {2}",  
    primerNumero, segundoNumero, suma);
```

Ya sabemos todo lo suficiente para crear nuestro programa que sume dos números usando variables:

```
public class Ejemplo02
{
    public static void Main()
    {
        int primerNumero;
        int segundoNumero;
        int suma;

        primerNumero = 234;
        segundoNumero = 567;
        suma = primerNumero + segundoNumero;

        System.Console.WriteLine("La suma de {0} y {1} es {2}",
            primerNumero, segundoNumero, suma);
    }
}
```

RECORDEMOS COMO SE DECLARAN LAS VARIABLES DE TIPO INT

Crea un programa que calcule el producto de los números 121 y 132, usando variables.

Crea un programa que calcule la suma de 285 y 1396, usando variables.

Crea un programa que calcule el resto de dividir 3784 entre 16, usando variables.

MANOS A LA OBRA TERCER PRODUCTO

Nombre del producto: Analogía de variables

Nomenclatura: MI-SII Analogía de variables

Tema: Variables int

Duración: 4 Sesiones

Objetivos: *identificar las características de las variables que se utilizan en la programación con C# para emplearlas en la confección de un programa*

Materiales y equipo: Libreta, hojas blancas, lapiceros, objetos necesarios.

Procedimiento:



1. Se presentan alguna forma de inicios de analogías en donde se pueden guardar diferentes artículos o herramientas, desglosa la funcionalidad principal en los casos de una variable escribe en los espacios en blanco analogías de la vida real

1. Analogía de una variable que guarda solo cereales	2. Analogía de una variable que almacena líquidos	3. Analogía de una variable que almacena frutas	4.
5.	6.	7.	8.
9.	10.	11.	12.

Nombre del producto:				
Nombre del alumno:				
Indicadores	Muy bien	Bien	Regular	Deficiente
	2.5	2	1.5	1
Título de la analogía				
Ejemplifica de forma correcta como almacena				
Ejemplifica las tareas relacionadas con un programa				
Expresa un buen proceso de lo representado				
Tienen presentación adecuada y formal el producto				
Participa de forma activa en clase				
Nombre y firma del docente	Nombre y firma del padre tutor			

A LO QUE LLEGAMOS

Menciona la importancia de leer bien las instrucciones para comprender lo que pide un problema a resolver por medio de un programa.

LO QUE SABES SOBRE UN IDENTIFICADOR



Un identificador es el nombre que se asigna a un tipo (clase, interfaz, estructura, registro, delegado o enumeración), miembro, variable o espacio de nombres.

PARA EMPEZAR IDENTIFICADORES

Identificadores

Estos nombres de variable (lo que se conoce como "identificadores") pueden estar formados por letras, números o el símbolo de subrayado (_) y deben comenzar por letra o subrayado. No deben tener espacios entre medias, y hay que recordar que las vocales acentuadas y la ñe son problemáticas, porque no son letras "estándar" en todos los idiomas.

Por eso, no son nombres de variable válidos:

1numero	(empieza por número)
un numero	(contiene un espacio)
Año1	(tiene una eñe)
MásDatos	(tiene una vocal acentuada)

Tampoco podremos usar como identificadores las **palabras reservadas** de C#. Por ejemplo, la palabra "int" se refiere a que cierta variable guardará un número entero, así que esa palabra "int" no la podremos usar tampoco como nombre de variable (pero no vamos a incluir ahora una lista de palabras reservadas de C#, ya nos iremos encontrando con ellas).

De momento, intentaremos usar nombres de variables que a nosotros nos resulten claros, y que no parezca que puedan ser alguna orden de C#.

Hay que recordar que en C# las **mayúsculas y minúsculas** se consideran diferentes, de modo que si intentamos hacer `PrimerNumero = 0;` `primernumero = 0;` o cualquier variación similar, el compilador protestará y nos dirá que no conoce esa variable, porque la habíamos declarado como `int primerNumero;`

```
PrimerNumero = 0;  
primernumero = 0;
```

o cualquier variación similar, el compilador protestará y nos dirá que no conoce esa variable, porque la habíamos declarado como

```
int primerNumero;
```

Los identificadores válidos deben seguir estas reglas:

- Los identificadores deben comenzar con una letra o un carácter de subrayado (`_`).
- Los identificadores pueden contener caracteres de letra Unicode, caracteres de dígito decimales, caracteres de conexión Unicode, caracteres de combinación Unicode o caracteres de formato Unicode. Para obtener más información sobre las categorías Unicode, vea la base de datos de categorías Unicode. Puede declarar identificadores que coincidan con palabras clave de C# mediante el

prefijo @ en el identificador. @ no forma parte del nombre de identificador. Por ejemplo, @if declara un identificador denominado if. Estos identificadores textuales son principalmente para la interoperabilidad con los identificadores declarados en otros lenguajes.

Comentarios

Podemos escribir comentarios, que el compilador ignora, pero que pueden servir para aclararnos cosas a nosotros. Se escriben entre /* y */:

```
int suma; /* Porque guardaré el valor para usarlo más tarde */
```

Es conveniente escribir comentarios que aclaren la misión de las partes de nuestros programas que puedan resultar menos claras a simple vista. Incluso suele ser aconsejable que el programa comience con un comentario, que nos recuerde qué hace el programa sin que necesitemos mirarlo de arriba a abajo. Un ejemplo casi exagerado:

```
/* ---- Ejemplo en C#: sumar dos números prefijados ---- */  
  
public class Ejemplo02b  
{  
    public static void Main()  
    {  
        int primerNumero = 234;  
        int segundoNumero = 567;  
        int suma; /* Guardaré el valor para usarlo más tarde */  
  
        /* Primero calculo la suma */  
        suma = primerNumero + segundoNumero;  
  
        /* Y después muestro su valor */  
        System.Console.WriteLine("La suma de {0} y {1} es {2}",  
            primerNumero, segundoNumero, suma);  
    }  
}
```

Un comentario puede empezar en una línea y terminar en otra distinta, así:

```
/* Esto  
es un comentario que  
ocupa más de una línea  
*/
```

También es posible declarar otro tipo de comentarios, que comienzan con doble barra y terminan cuando se acaba la línea (estos comentarios, claramente, no podrán ocupar más de una línea). Son los "comentarios al estilo de C++":

```
// Este es un comentario "al estilo C++"
```

RECORDEMOS IDENTIFICADORES Y COMENTARIOS

Ejecuta el programa visto en la sesión a través del compilador adecuado instalado en tu herramienta de trabajo.

Recuerda que de ahora en adelante todos tus programas deben de llevar tu nombre y datos indicados por el docente.

LO QUE SABES DEL MÉTODO INT 32



Int32.Parse lo vamos a utilizar cuando estamos completamente seguros que el texto corresponde a un valor numerico, por ejemplo, si tengo que convertir un valor proveniente de una caja de texto a un entero y hemos limitado a que sólo se ingresen números, no vamos a tener problemas con Int32.Parse.

PARA EMPEZAR A UTILIZAR READLINE

Datos por el usuario: ReadLine

Si queremos que sea el usuario de nuestro programa quien teclee los valores, necesitamos una nueva orden, que nos permita leer desde teclado. Pues bien, al igual que tenemos `System.Console.WriteLine` ("escribir línea"), también existe `System.Console.ReadLine` ("leer línea"). Para leer textos, haríamos

```
texto = System.Console.ReadLine();
```

pero eso ocurrirá en el próximo tema, cuando veamos cómo manejar textos. De momento, nosotros sólo sabemos manipular números enteros, así que deberemos convertir ese dato a un número entero, usando `Convert.ToInt32`:

```
primerNumero = System.Convert.ToInt32( System.Console.ReadLine() );
```

Un ejemplo de programa que sume dos números tecleados por el usuario sería:

```
public class Ejemplo03
{
    public static void Main()
    {
        int primerNumero;
        int segundoNumero;
        int suma;

        System.Console.WriteLine("Introduce el primer número");
        primerNumero = System.Convert.ToInt32(
            System.Console.ReadLine());
        System.Console.WriteLine("Introduce el segundo número");
        segundoNumero = System.Convert.ToInt32(
            System.Console.ReadLine());
        suma = primerNumero + segundoNumero;

        System.Console.WriteLine("La suma de {0} y {1} es {2}",
            primerNumero, segundoNumero, suma);
    }
}
```

Multiplicar dos números tecleados por usuario. El programa deberá contener un comentario al principio, que recuerde cual es su objetivo.

El usuario tecleará dos números (x e y), y el programa deberá calcular cuál es el resultado de su división y el resto de esa división. Deberás usar "Write" en vez de "WriteLine" para pedir los datos, e incluir un comentario con tu nombre y la fecha en que has realizado el programa.

El usuario tecleará dos números (a y b), y el programa mostrará el resultado de la operación $(a+b)*(a-b)$ y el resultado de la operación $a^2 - b^2$.

Sumar tres números tecleados por usuario.

Pedir al usuario un número y mostrar su tabla de multiplicar. Por ejemplo, si el número es el 3, debería escribirse algo como

$$3 \times 0 = 0$$

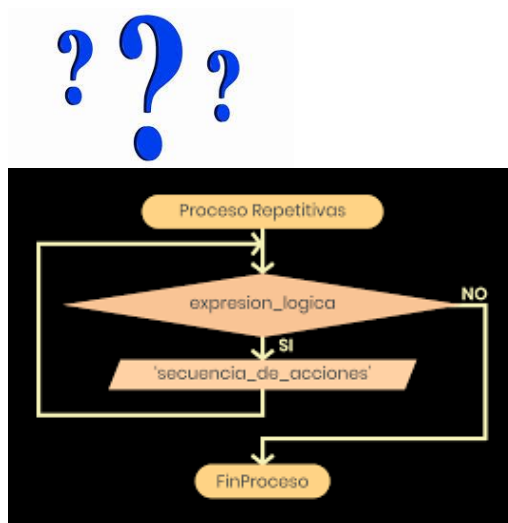
$$3 \times 1 = 3$$

$$3 \times 2 = 6$$

...

$$3 \times 10 = 30$$

LO QUE SABES DE



Los condicionales se utilizan para expresar probabilidades, propuestas, dudas o deseos. Por ejemplo: Si fuera más estudiosa, tendría mejores notas.

PARA EMPEZAR ESTRUCTURA DE CONTROL

La sentencia if

Vamos a ver cómo podemos comprobar si se cumplen condiciones. La primera construcción que usaremos será "si ... entonces ...". El formato es

```
if (condición) sentencia;

using System;

public class Ejemplo05
{
    public static void Main()
    {
        int numero;

        Console.WriteLine("Introduce un número");
        numero = Convert.ToInt32(Console.ReadLine());
        if (numero>0) Console.WriteLine("El número es positivo.");
    }
}
```

Este programa pide un número al usuario. Si es positivo (mayor que 0), escribe en pantalla "El número es positivo."; si es negativo o cero, no hace nada.

Como se ve en el ejemplo, para comprobar si un valor numérico es mayor que otro, usamos el símbolo ">". Para ver si dos valores son iguales, usaremos dos símbolos de "igual": `if (numero==0)`. Las demás posibilidades las veremos algo más adelante. En todos los casos, la condición que queremos comprobar deberá indicarse entre paréntesis.

Este programa comienza por un comentario que nos recuerda de qué se trata. Como nuestros fuentes irán siendo cada vez más complejos, a partir de ahora incluiremos comentarios que nos permitan recordar de un vistazo qué pretendíamos hacer.

Si la orden "if" es larga, se puede partir en dos líneas para que resulte más legible:

```
if (numero>0)
    Console.WriteLine("El número es positivo.");
```

RECORDEMOS COMO FUNCIONA EL IF

I. Ejecuta el siguiente programa para comprender el uso de if simple.

```
using System;

public class Ejemplo05
{
    public static void Main()
    {
        int numero;

        Console.WriteLine("Introduce un número");
        numero = Convert.ToInt32(Console.ReadLine());
        if (numero>0) Console.WriteLine("El número es positivo.");
    }
}
```

MANOS A LA OBRA CON EL CUARTO PRODUCTO

Nombre de la práctica: Estructura de control IF

Nomenclatura: MI-SII Practica I/Estructura IF

Tema: Estructuras de control IF

Duración: 5 Sesiones

Objetivos: *utilizar las estructuras de control para la toma de decisiones, con ayuda de la programación*

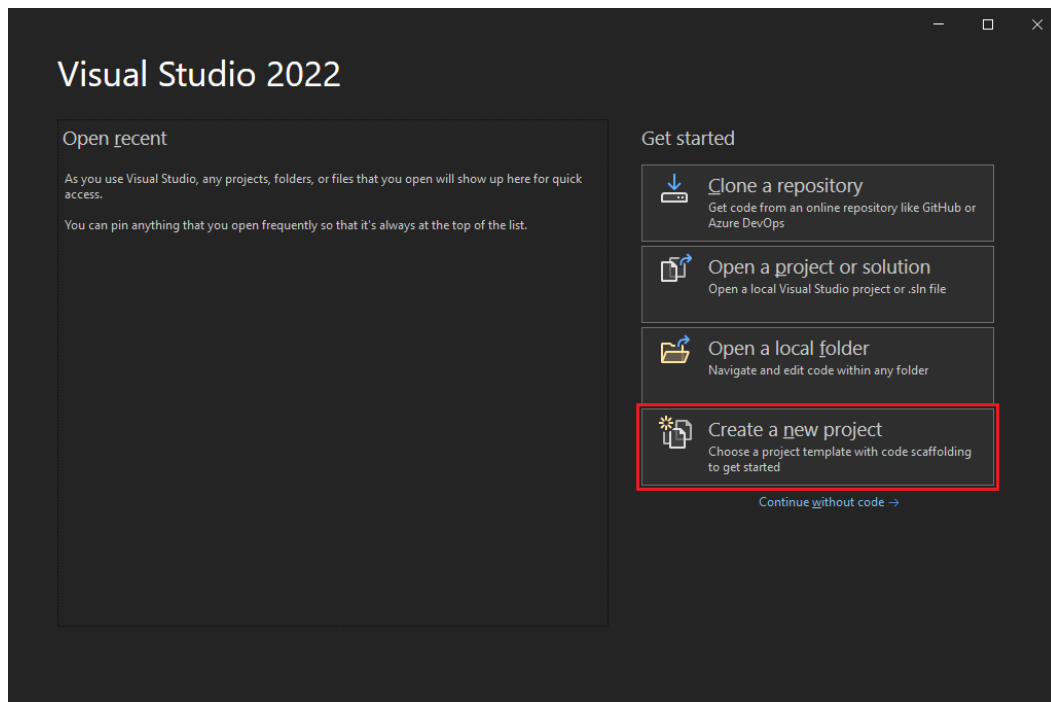
Materiales y equipo: Equipo de cómputo con software instalado Visual Studio o en su caso aplicación de Play Store

Procedimiento:

Crear un proyecto

Para empezar, cree un proyecto de aplicación de C#. En el tipo de proyecto se incluyen todos los archivos de plantilla que necesita.

Abra Visual Studio y elija **Crear un nuevo proyecto** en la ventana Inicio.



- En la ventana **Crear un nuevo proyecto**, seleccione **Todos los lenguajes** y, a continuación, elija **C#** en la lista desplegable. Seleccione **Windows** en la lista **Todas las plataformas** y **Consola** en la lista **Todos los tipos de proyecto**.

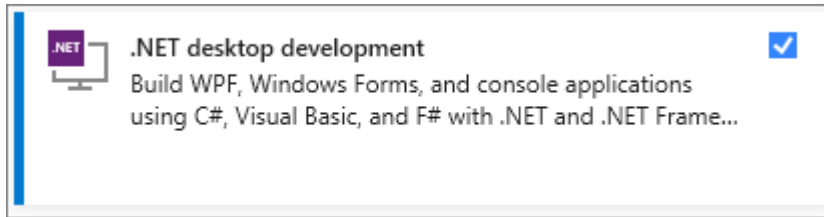
Después de aplicar los filtros de lenguaje, plataforma y tipo de proyecto, elija la plantilla **Aplicación de consola** y, luego, seleccione **Siguiente**.

Nota

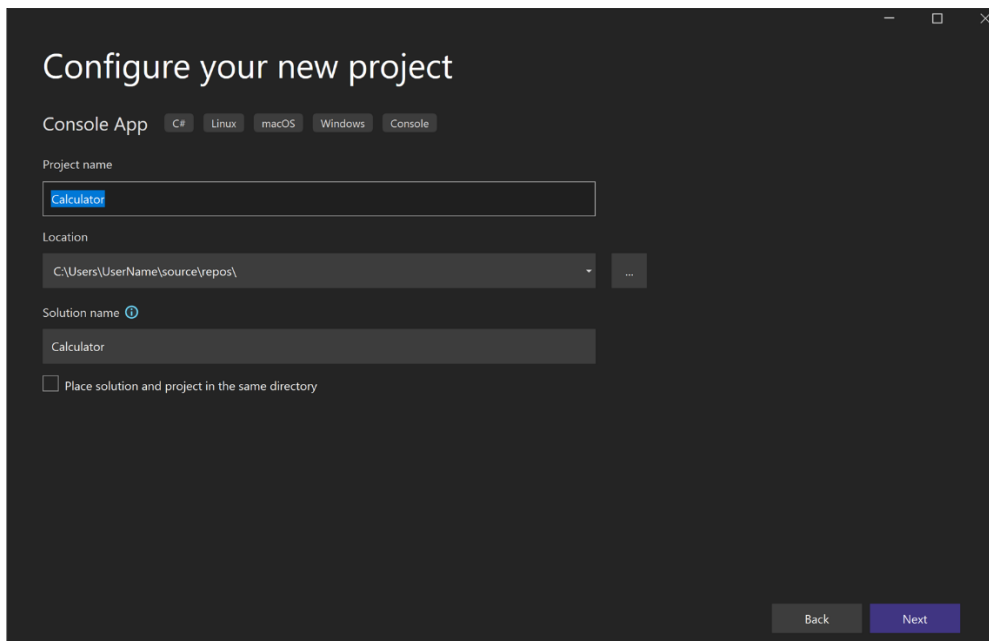
Si no ve la plantilla **Aplicación de consola**, seleccione **Instalar más herramientas y características**.

Not finding what you're looking for?
[Install more tools and features](#)

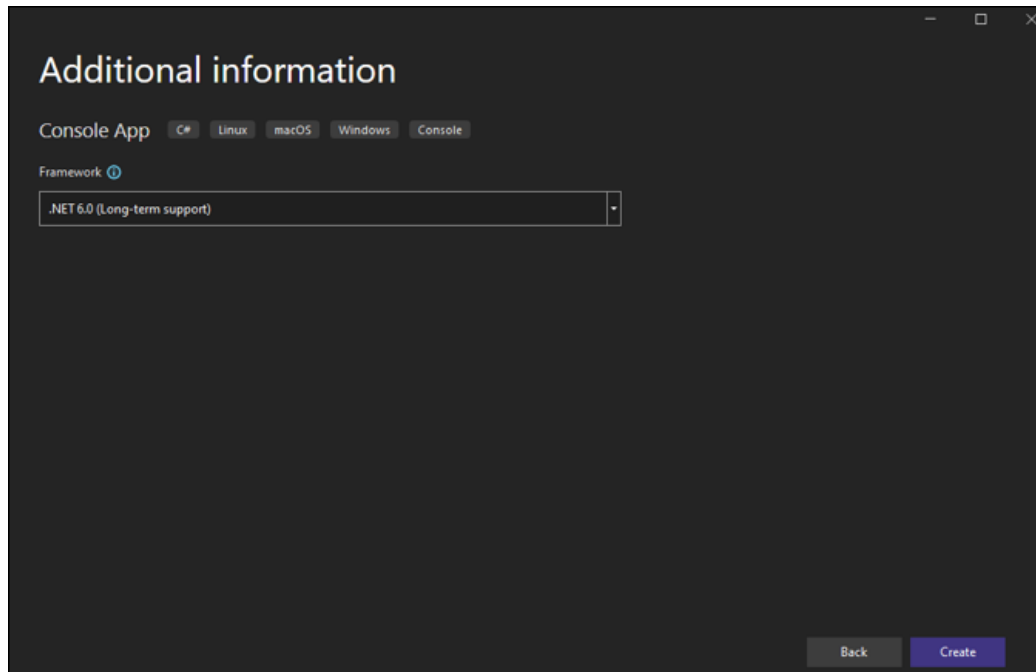
En el Instalador de Visual Studio, elija la carga de trabajo de **desarrollo de escritorio de .NET** y, a continuación, seleccione **Modificar**.



9. En la ventana **Configurar el nuevo proyecto**, escriba CALCULATOR en el cuadro **Nombre del proyecto** y, a continuación, seleccione **Siguiente**.



10. En la ventana **Información adicional**, **.NET 6.0** ya debe estar seleccionado para la plataforma de destino. Seleccione **Crear**.



Visual Studio abre el nuevo proyecto, que incluye código predeterminado de "Hola mundo".

Para verlo en el editor, seleccione el archivo de código Program.cs en la ventana del Explorador de soluciones, que normalmente se encuentra en el lado derecho de Visual Studio.

La instrucción de código única llama al método WriteLine para mostrar la cadena literal "¡Hola mundo!" en la ventana de consola. Si presiona F5, puede ejecutar el programa predeterminado en modo de depuración. Una vez que la aplicación se ejecuta en el depurador, la ventana de la consola permanece abierta. Presione cualquier tecla para cerrar la ventana de consola.

Nota

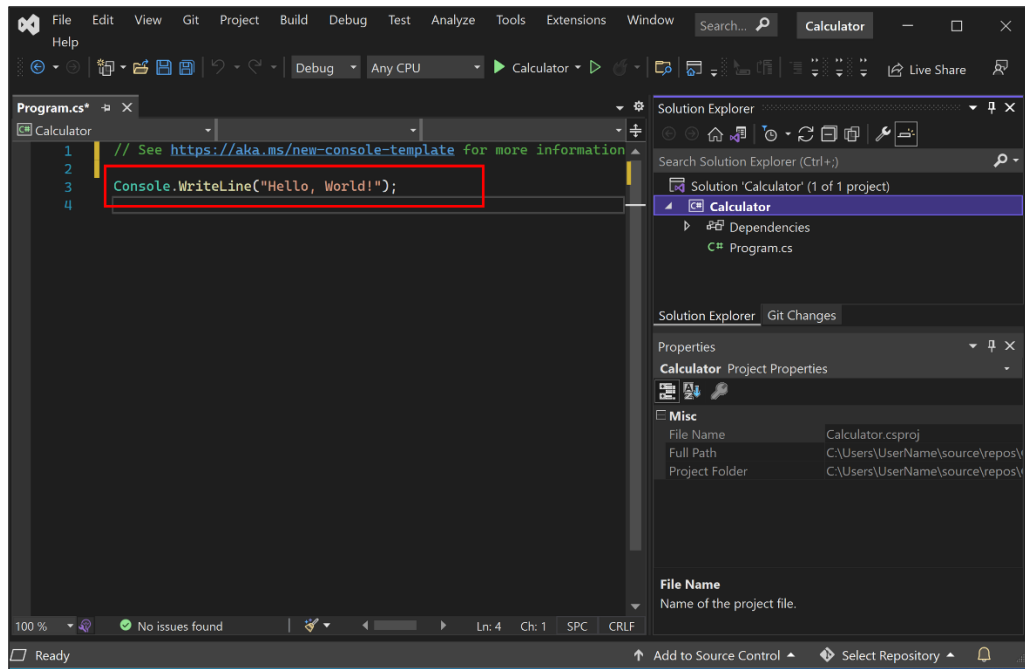
A partir de .NET 6, los nuevos proyectos que usan la plantilla de consola generan código diferente al de versiones anteriores. Para obtener más información, consulte la página Las nuevas plantillas de C# generan instrucciones de nivel superior.

Creación de la aplicación

Empiece con algunos cálculos básicos de enteros en C#.

11. En el **Explorador de soluciones**, en el panel derecho, seleccione **Program.cs** para mostrar el archivo en el editor de código

12. En el editor de código, reemplace el código predeterminado "Hello World" que dice `Console.WriteLine("Hello World!");`;



13. Crear un programa que pida al usuario un número entero. Si es múltiplo de 10, se lo avisará al usuario



Aspectos a evaluar	Ponderación		
	Porcentaje	sí	No
Portada con los datos de identificación	2		
Código del programa que pida al usuario un entero y diga si es multiplo de 10	4		
Resultado del programa que pida al usuario un entero y diga si es multiplo de 10	4		
Resultado del programa que calcule el resultado de dividir 301 entre 3.	2		
Tiene limpieza y formalidad	2		
Se entrega con un folder en condiciones optimas de un trabajo de media superior	1		
Nombre y firma del docente	Nombre y firma del padre tutor		



SEGUNDO PARCIAL

ENCUADRE

COMPETENCIA PROFESIONAL

Reconoce las características de un lenguaje de programación estructurado

Desarrolla código en un lenguaje de programación estructurado

SITUACIÓN DE APRENDIZAJE



PLAN DE EVALUACIÓN

PRODUCTOS DE APRENDIZAJE SEGUNDO PARCIAL

PRODUCTO	PONDERACIÓN	PORCENTAJE OBTENIDO	OBSERVACIONES
Proyecto integrador IF-ELSE, WHILE, DO-WHILE	30%		
Modulo de programas	30%		
Programa de diferencia de valores	20%		
Tutorías y Orientación Educativa	10%		
Innovación y liderazgo CECyTE	10%		

INICIO

Aprendizajes claves

LO QUE SABES DE LOS CONDICIONALES



Los condicionales se utilizan para expresar probabilidades, propuestas, dudas o deseos. Por ejemplo: Si fuera más estudiosa, tendría mejores notas.

PARA EMPEZAR ESTRUCTURA CONDICIONAL IF – ELSE

Condicional if – else

La declaración if verifica una condición y ejecuta un bloque si la condición es verdadera.

Y la instrucción if else verifica una condición y ejecuta un bloque si la condición es verdadera u otro bloque en caso contrario.

Tanto las declaraciones if como if else verifican solo una condición.

A veces, es posible que desees verificar varias condiciones y ejecutar un bloque si una condición es verdadera. Para hacer eso, puedes usar la declaración if else if.

Esta es la sintaxis de la instrucción if else if:



```
if (condition1)
{
    // block 1
}
else if (condition2)
{
    // block 2
}
else if (condition3)
{
    // block 3
}
else
{
    // else block
}
```

Ejemplo

```
using System;

public class Ejemplo08
{
    public static void Main()
    {
        int numero;

        Console.WriteLine("Introduce un número");
        numero = Convert.ToInt32(Console.ReadLine());
        if (numero > 0)
            Console.WriteLine("El número es positivo.");
        else
            Console.WriteLine("El número es cero o negativo.");
    }
}
```

Podríamos intentar evitar el uso de "else" si utilizamos un "if" a continuación de otro, así:

```
using System;

public class Ejemplo09
{
    public static void Main()
    {
        int numero;

        Console.WriteLine("Introduce un número");
        numero = Convert.ToInt32(Console.ReadLine());

        if (numero > 0)
            Console.WriteLine("El número es positivo.");

        if (numero <= 0)
            Console.WriteLine("El número es cero o negativo.");
    }
}
```

Pero el comportamiento no es el mismo: en el primer caso (ejemplo 8) se mira si el valor es positivo; si no lo es, se pasa a la segunda orden, pero si lo es, el programa ya ha terminado. En el segundo caso (ejemplo 9), aunque el número sea positivo, se vuelve a realizar la segunda comprobación para ver si es negativo o cero, por lo que el programa es algo más lento.

Podemos enlazar varios "if" usando "else", para decir "si no se cumple esta condición, mira a ver si se cumple esta otra":

```
using System;

public class Ejemplo10
{
    public static void Main()
    {
        int numero;

        Console.WriteLine("Introduce un número");
        numero = Convert.ToInt32(Console.ReadLine());

        if (numero > 0)
            Console.WriteLine("El número es positivo.");
    }
}
```



```
else
    if (numero < 0)
        Console.WriteLine("El número es negativo.");
    else
        Console.WriteLine("El número es cero.");
}
```

RECORDEMOS EL USO DEL IF – ELSE

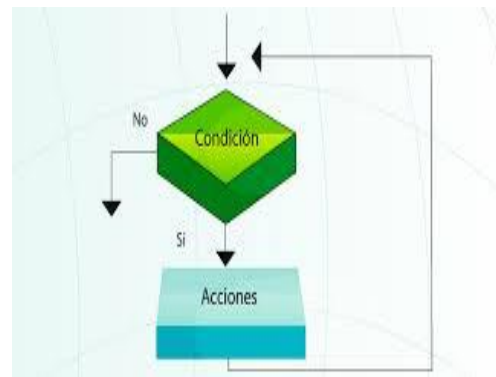
Confecciona los siguientes programas

1. Cree un programa que permita saber, en base a la nota final del curso de un alumno, si ha aprobado o ha desaprobado. Si la nota es mayor que 5, el alumno ha aprobado, en caso contrario, ha desaprobado.
2. Cree un programa que valide si el número ingresado es par o impar.

LO QUE SABES DE ESTRUCTURAS REPETITIVAS

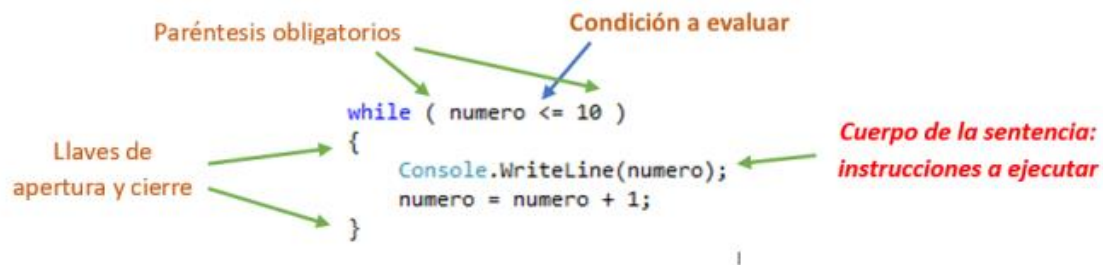


Una estructura repetitiva o bucle se utiliza cuando se quiere repetir un conjunto de sentencias un número determinado de veces o mientras se mantenga el cumplimiento de una condición.



PARA EMPEZAR ESTRUCTURA REPETITIVA WHILE - MIENTRAS

While es una estructura de control cuyo propósito es ejecutar un bloque de código que consiste en repetir una instrucción mientras se cumpla cierta condición argumentada dentro de los paréntesis, su sintaxis es la siguiente;



Si queremos hacer que una sección de nuestro programa se repita mientras se cumpla una cierta condición, usaremos la orden "while". Esta orden tiene dos formatos distintos, según comprobemos la condición al principio o al final.

En el primer caso, su sintaxis es

```
while (condición)  
    sentencia;
```

Es decir, la sentencia se repetirá mientras la condición sea cierta. Si la condición es falsa ya desde un principio, la sentencia no se ejecuta nunca. Si queremos que se repita más de una sentencia, basta agruparlas entre { y }.

Un ejemplo que nos diga si cada número que tecleemos es positivo o negativo, y que pare cuando tecleemos el número 0, podría ser:



```
using System;

public class Ejemplo16
{
    public static void Main()
    {
        int numero;

        Console.WriteLine("Teclea un número (0 para salir): ");
        numero = Convert.ToInt32(Console.ReadLine());

        while (numero != 0)
        {
            if (numero > 0) Console.WriteLine("Es positivo");
            else Console.WriteLine("Es negativo");

            Console.WriteLine("Teclea otro número (0 para salir): ");
            numero = Convert.ToInt32(Console.ReadLine());
        }
    }
}
```

En este ejemplo, si se introduce 0 la primera vez, la condición es falsa y ni siquiera se entra al bloque del "while", terminando el programa inmediatamente.

Ahora que sabemos "repetir" cosas, podemos utilizarlo también para contar. Por ejemplo, si queremos contar del 1 al 5, nuestra variable empezaría en 1, aumentaría una unidad en cada repetición y se repetiría hasta llegar al valor 5, así:

```
using System;

public class Ejemplo16b
{
    public static void Main()
    {
        int n = 1;

        while (n < 6)
        {
            Console.WriteLine(n);
            n = n + 1;
        }
    }
}
```

RECORDEMOS EL USO DE WHILE

Realiza el siguiente compendio de programas

1. Realiza un programa en C#, que muestre los primeros 100 números de forma inversa, es decir, del 100 al 1
2. Realiza un programa en C#, que muestre únicamente, los números pares en el rango del 1 al 100
3. Escribir un programa que solicite la carga de números por teclado, obtener su promedio. Finalizar la carga de valores cuando se cargue el valor 0.

Cuando la finalización depende de algún valor ingresado por el operador conviene el empleo de la estructura do while, por lo menos se cargará un valor (en el caso más extremo se carga 0, que indica la finalización de la carga de valores)

LO QUE SABES DE BUCLES





Un bucle o ciclo, en programación, es una secuencia de instrucciones de código que se ejecuta repetidas veces, hasta que la condición asignada a dicho bucle ...

PARA EMPEZAR ESTRUCTURAS DE CONTROL DO ... WHILE

Estructura do – while

Este es el otro formato que puede tener la orden "while": la condición se comprueba al final (equivale a "repetir...mientras"). El punto en que comienza a repetirse se indica con la orden "do", así:

```
do  
    sentencia;  
while (condición)
```

La única diferencia entre While y Do-While, es que la condición se traslada hacia la parte final del bloque de código. Es decir, que siempre se va a ejecutar el bucle al menos una vez, y luego se evalúa la condición para saber si se continua ejecutando. En caso la condición sea falsa, el bloque de código Do-While termina y continua ejecutando el código posterior. Veamos un ejemplo.

vamos a ver cómo sería el típico programa que nos pide una clave de acceso y no nos deja entrar hasta que tecleemos la clave correcta:



```
using System;

public class Ejemplo17
{
    public static void Main()
    {
        int valida = 711;
        int clave;

        do
        {
            Console.Write("Introduzca su clave numérica: ");
            clave = Convert.ToInt32(Console.ReadLine());

            if (clave != valida)
                Console.WriteLine("No válida!");
        }
        while (clave != valida);

        Console.WriteLine("Aceptada.");
    }
}
```

En este caso, se comprueba la condición al final, de modo que se nos preguntará la clave al menos una vez. Mientras que la respuesta que demos no sea la correcta, se nos vuelve a preguntar. Finalmente, cuando tecleamos la clave correcta, el ordenador escribe "Aceptada" y termina el programa.

Como veremos un poco más adelante, si preferimos que la clave sea un texto en vez de un número, los cambios al programa son mínimos, basta con usar "string":

```
using System;

public class Ejemplo18
{
    public static void Main()
    {
        string valida = "secreto";
        string clave;

        do
        {
            Console.Write("Introduzca su clave: ");
            clave = Console.ReadLine();

            if (clave != valida)
                Console.WriteLine("No válida!");
        }
        while (clave != valida);

        Console.WriteLine("Aceptada.");
    }
}
```

RECORDA EL EMPLEO DE DO WHILE

Actividad

Confecciona los siguientes programas

Realiza un programa que pida N sueldos, los sume y al final de el promedio de todos los sueldos

MANOS A LA OBRA CON PRIMER PRODUCTO SEGUNDO PARCIAL

Nombre de la práctica: Proyecto integrador

Nomenclatura: MI-SII Practica I/estructura IF-ELSE, WHILE, DO WHILE

Tema: Estructuras de control IF-ELSE, WHILE, DO WHILE

Duración: 8 Sesiones

Objetivos: *utilizar las estructuras de control para la toma de decisiones, empleando las diferentes sentencias en el caso de tener alternativas de elección, para la solución de problemas*

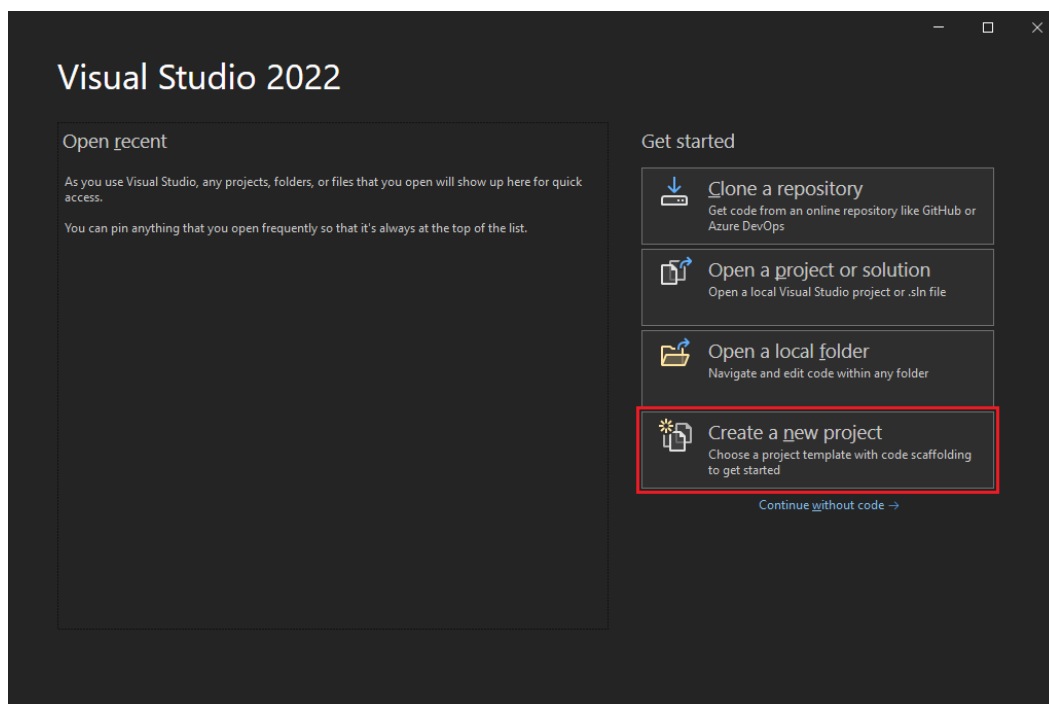
Materiales y equipo: Equipo de cómputo con software instalado Visual Studio o en su caso aplicación de Play Store

Procedimiento:

Crear un proyecto

Para empezar, cree un proyecto de aplicación de C#. En el tipo de proyecto se incluyen todos los archivos de plantilla que necesita.

Abra Visual Studio y elija **Crear un nuevo proyecto** en la ventana Inicio.



1. En la ventana **Crear un nuevo proyecto**, seleccione **Todos los lenguajes** y, a continuación, elija **C#** en la lista desplegable. Seleccione **Windows** en la lista **Todas las plataformas** y **Consola** en la lista **Todos los tipos de proyecto**.

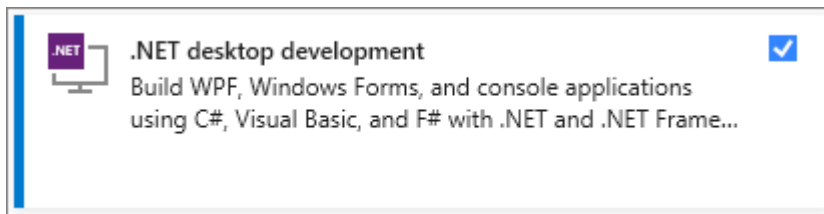
Después de aplicar los filtros de lenguaje, plataforma y tipo de proyecto, elija la plantilla **Aplicación de consola** y, luego, seleccione **Siguiente**.

Nota

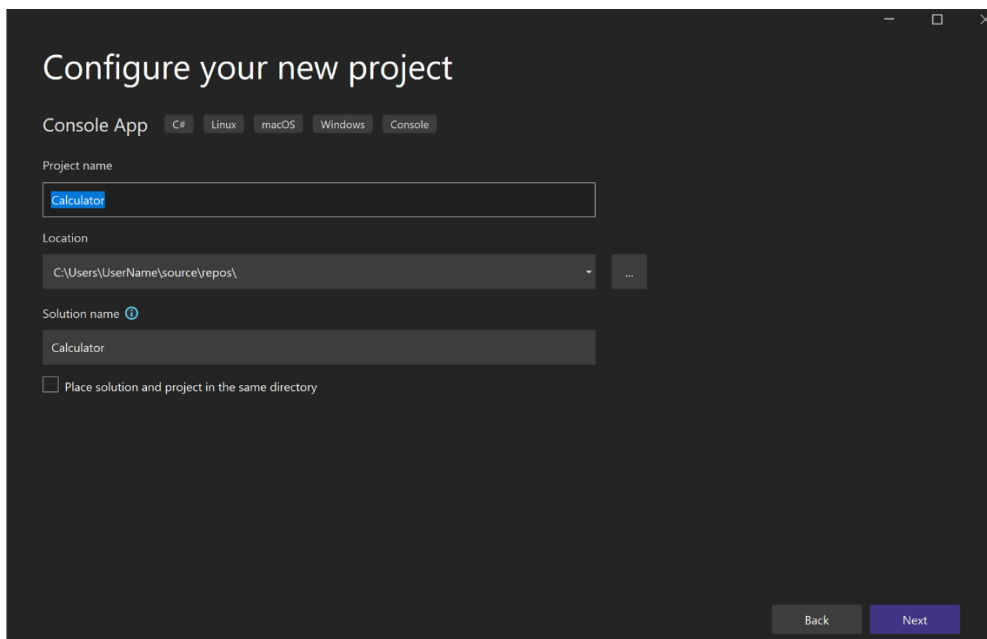
Si no ve la plantilla **Aplicación de consola**, seleccione **Instalar más herramientas y características**.

Not finding what you're looking for?
[Install more tools and features](#)

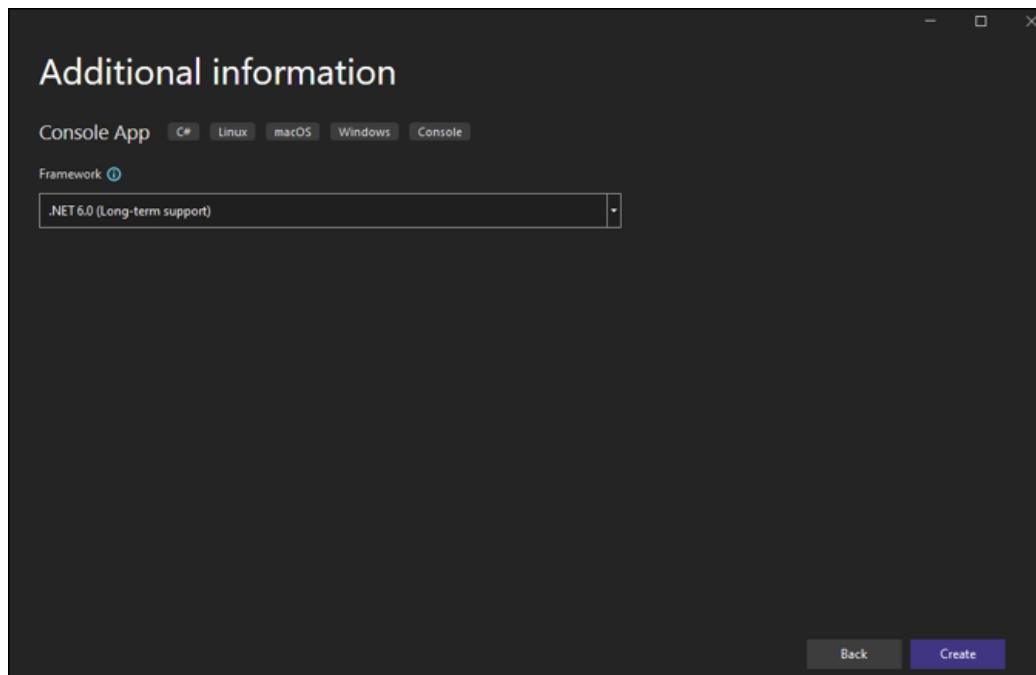
En el Instalador de Visual Studio, elija la carga de trabajo de **desarrollo de escritorio de .NET** y, a continuación, seleccione **Modificar**.



2. En la ventana **Configurar el nuevo proyecto**, escriba CALCULATOR en el cuadro **Nombre del proyecto** y, a continuación, seleccione **Siguiente**.



3. En la ventana **Información adicional**, **.NET 6.0** ya debe estar seleccionado para la plataforma de destino. Seleccione **Crear**.



Visual Studio abre el nuevo proyecto, que incluye código predeterminado de "Hola mundo".

Para verlo en el editor, seleccione el archivo de código Program.cs en la ventana del Explorador de soluciones, que normalmente se encuentra en el lado derecho de Visual Studio.

La instrucción de código única llama al método WriteLine para mostrar la cadena literal "¡Hola mundo!" en la ventana de consola. Si presiona F5, puede ejecutar el programa predeterminado en modo de depuración. Una vez que la aplicación se ejecuta en el depurador, la ventana de la consola permanece abierta. Presione cualquier tecla para cerrar la ventana de consola.

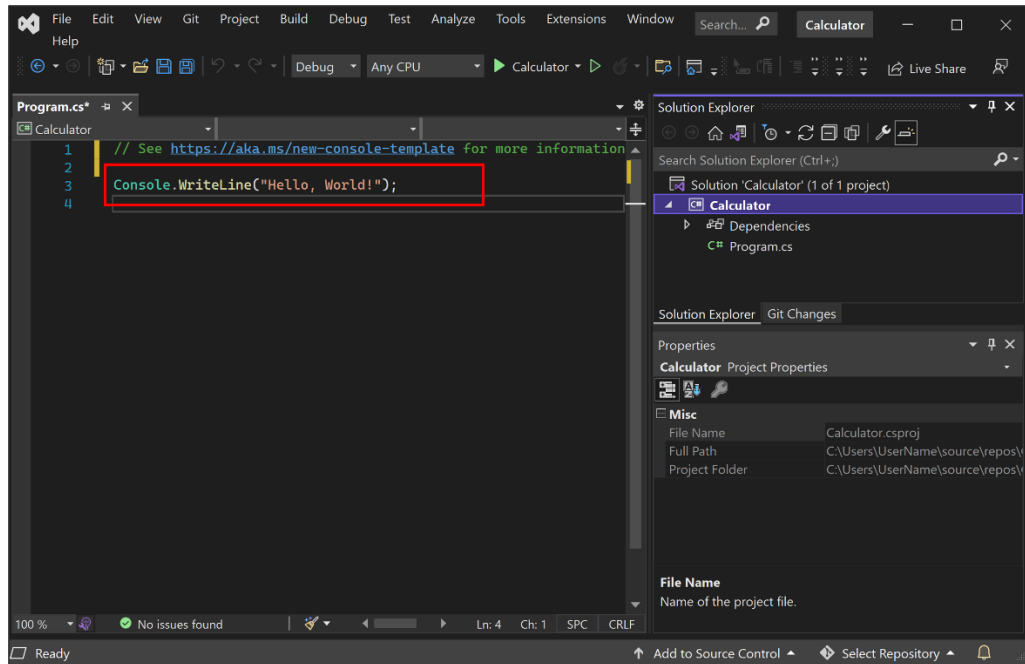
Nota

A partir de .NET 6, los nuevos proyectos que usan la plantilla de consola generan código diferente al de versiones anteriores. Para obtener más información, consulte la página Las nuevas plantillas de C# generan instrucciones de nivel superior.

Creación de la aplicación

Empiece con algunos cálculos básicos de enteros en C#.

4. En el **Explorador de soluciones**, en el panel derecho, seleccione **Program.cs** para mostrar el archivo en el editor de código
5. En el editor de código, reemplace el código predeterminado "Hello World" que dice `Console.WriteLine("Hello World!");`;



6. Ejecuta los siguientes bloques de programas

Bloque de IF-ELSE

- Cree un programa que permita saber, en base a la **nota final del curso de un alumno**, si ha aprobado o ha desaprobado. Si la nota es mayor que 5, el alumno ha aprobado, en caso contrario, ha desaprobado.
- Cree un programa que **valide si el número ingresado es par o impar**.



Bloque de WHILE

- Realiza un programa en C#, que muestre los primeros **100 números de forma inversa, es decir, del 100 al 1**
- Realiza un programa en C#, que muestre únicamente, los **números pares en el rango del 1 al 100**

Bloque de DO-WHILE

- Realiza un programa que pida **N sueldos, los sume y al final de el promedio de todos los sueldos**

Aspectos a evaluar	Ponderación		
	Porcentaje	sí	No
Portada con los datos de identificación	2		
Código del Programa que valide si el número ingresado es par o impar	3		
Resultado del Programa que valide si el número ingresado es par o impar	3		
Código del programa que muestre 100 números de forma inversa, es decir, del 100 al 1	3		
Resultado del programa que muestre 100 números de forma inversa, es decir, del 100 al 1	3		
Código del programa que muestre números pares en el rango del 1 al 100	3		
Resultado del programa que muestre números pares en el rango del 1 al 100	3		
Código del programa que pida N sueldos, los sume y al final del promedio de todos los sueldos	3		
Resultado del programa que pida N sueldos, los sume y al final del promedio de todos los sueldos	3		
Tiene limpieza y formalidad	2		
Se entrega con un folder en condiciones optimas de un trabajo de media superior	2		
Nombre y firma del docente	Nombre y firma del padre tutor		

LO QUE SABES SOBRE LOS TIPOS DE DATOS



Tipos de datos: los tipos de datos en programación se distinguen numéricos, textos y lógicos.

PARA EMPEZAR TIPOS DE DATOS

¿Qué son los tipos de Datos?

Una variable contiene datos de un tipo específico. Cuando declaramos una variable para almacenar los datos en una aplicación, debemos elegir un tipo de dato adecuado para los datos. Visual C# es un lenguaje de tipos seguros “Type-Safe”, esto significa que el compilador garantiza que los valores almacenados en las variables siempre son del tipo apropiado. La siguiente tabla muestra los tipos de datos más comunes que son utilizados en Visual C#.

C# proporciona **16** tipos de datos **integrados** o **predefinidos** para representar números enteros, decimales, de punto flotante, valores booleanos, caracteres, cadenas de caracteres, y otros tipos. De estos, **13** se denominan **simples**.

Los **13** tipos predefinidos simples incluyen:

Once tipos numéricos:

- **Ocho** tipos enteros de varias longitudes, con y sin signo: `sbyte`, `byte`, `short`, `ushort`, `int`, `uint`, `long` y `ulong`.
- **Dos** tipos de punto flotante: `float` y `double`.

- **Un** tipo de mayor precisión llamado `decimal`, que a diferencia de `float` y `double`, puede representar números con fracciones exactas. Lo que lo hace adecuado para cálculos financieros, monetarios, operaciones aritméticas, etc.
- **Un** tipo de carácter unicode, llamado `char`.
- `bool`, **Un** tipo que representa dos valores, verdadero y falso.

Los **3** tipos restantes o no simples son:

- `object`, que es el tipo base de todos los demás tipos.
- `string`, el cual representa un arreglo de caracteres Unicode.
- `dynamic`, el cual es usado para escribir assemblies en lenguajes dinámicos.

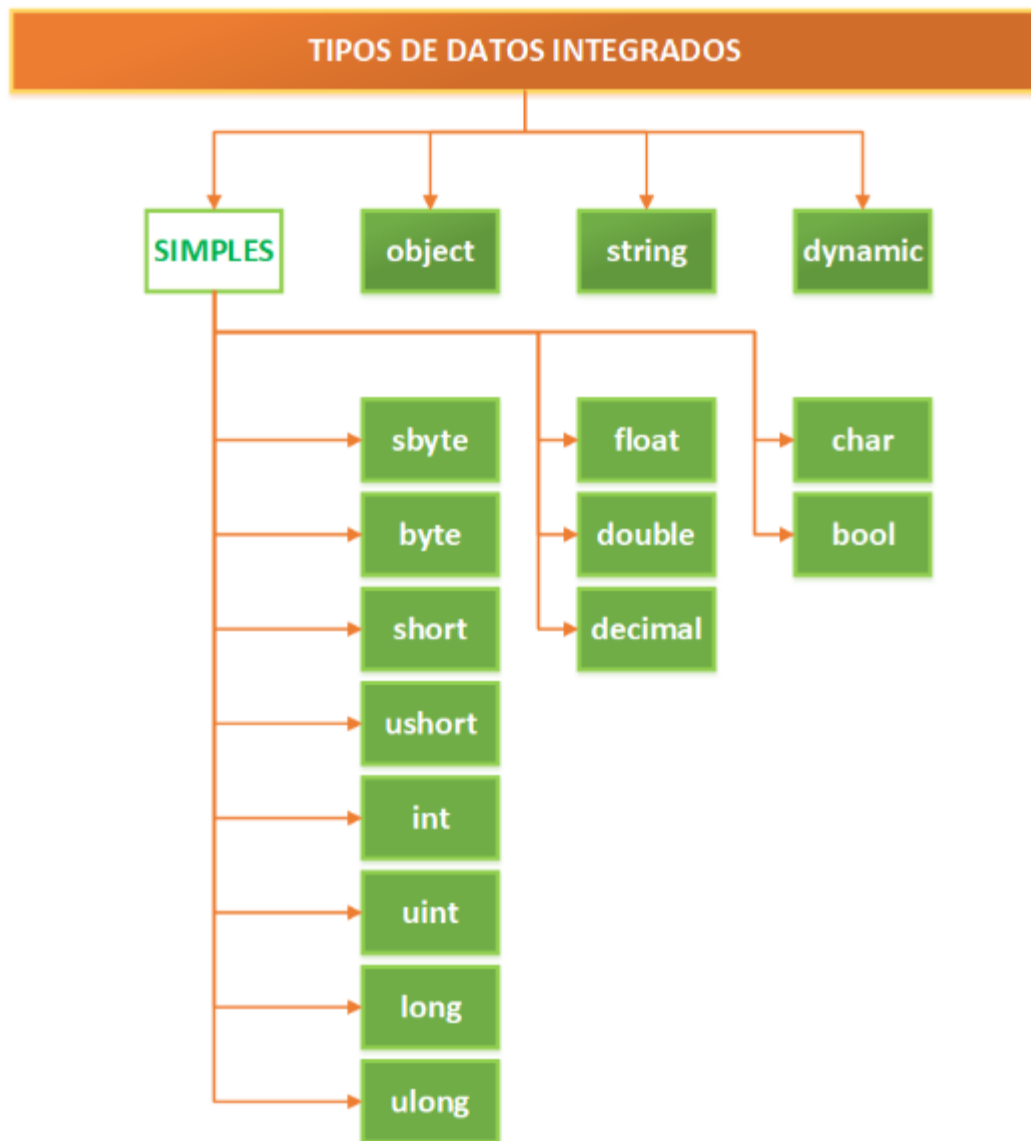


Tabla de Tipos Integrados Simples

Todos los tipos integrados simples del lenguaje C#, son alias de tipos .NET Framework predefinidos en el espacio de nombres `System`. Por ejemplo, `int` es un alias de `System.Int32`.

La siguiente tabla muestra todos los tipos de datos integrados simples.

Tipo C#	Intervalo	Tamaño / Precisión	Tipo .NET	Default
<code>sbyte</code>	De -128 a 127	Entero de 8 bits con signo	<code>System.SByte</code>	0
<code>byte</code>	De 0 a 255	Entero de 8 bits sin signo	<code>System.Byte</code>	0
<code>short</code>	De -32 768 a 32 767	Entero de 16 bits con signo	<code>System.Int16</code>	0
<code>ushort</code>	De 0 a 65,535	Entero de 16 bits sin signo	<code>System.UInt16</code>	0
<code>int</code>	De -2.147.483.648 a 2.147.483.647	Entero de 32 bits con signo	<code>System.Int32</code>	0
<code>uint</code>	De 0 a 4.294.967.295	Entero de 32 bits sin signo	<code>System.UInt32</code>	0
<code>long</code>	De -9.223.372.036.854.775.808 a 9.223.372.036.854.775.807	Entero de 64 bits con signo	<code>System.Int64</code>	0
<code>ulong</code>	De 0 a 18.446.744.073.709.551.615	Entero de 64 bits sin signo	<code>System.UInt64</code>	0
<code>float</code>	De $\pm 1,5 \times 10^{-45}$ a $\pm 3,4 \times 10^{38}$	7 dígitos	<code>System.Single</code>	0.0f
<code>double</code>	De $\pm 5,0 \times 10^{-324}$ a $\pm 1,7 \times 10^{308}$	15-16 dígitos	<code>System.Double</code>	0.0d
<code>decimal</code>	De $\pm 1,0 \times 10^{-28}$ to $\pm 7,9228 \times 10^{28}$	28-29 dígitos significativos	<code>System.Decimal</code>	0m
<code>char</code>	U+0000 a U+FFFF	Carácter Unicode de 16 bits	<code>System.Char</code>	\x0000
<code>bool</code>	Booleano	true, false	<code>System.Boolean</code>	false

Tabla de Tipos Integrados No Simples

Los tipos de datos integrados no simples son:

Tipo C#	Descripción	Tipo .NET
<code>object</code>	Es la clase base para todos los demás tipos, incluidos los tipos integrados simples.	<code>System.Object</code>
<code>string</code>	Una secuencia de caracteres Unicode.	<code>System.String</code>
<code>dynamic</code>	Es un tipo diseñado para ser usado con assemblies escritos en lenguajes dinámicos	No corresponde a un tipo .NET

RECORDEMOS USO DE TIPOS DE DATOS

Actividad

I. Imprime la coronilla de tipos de datos, recorta para ajustarla, para pegarla en la libreta y tenerla a la mano para las próximas utilidades en la confección de programas

LO QUE SABES DE CADENA DE CARACTERES

Cadena de caracteres

En computación, una cadena de caracteres o cadena de texto o simplemente cadena (`string` en inglés) es una secuencia ordenada de símbolos, con una longitud arbitraria (con tantos símbolos como queramos).

Se llama cadena, haciendo la analogía con una cadena física creada por elementos llamados eslabones, donde cada eslabón dentro de la cadena se encuentra acomodado en una secuencia consecutiva, uno detrás de otro. Como las cadenas son una secuencia ordenada de valores unos seguidos de otros, podemos hacer referencia a la posición de cada símbolo dentro de la cadena por medio de un número o índice, hay que tener en cuenta que en computación los índices generalmente se consideran desde la posición 0 y no desde el 1



PARA EMPEZAR

En C# hemos visto que cuando queremos almacenar un valor entero definimos una variable de tipo `int`, si queremos almacenar un valor con decimales definimos una variable de tipo `float`. Ahora si queremos almacenar una cadena de caracteres (por ejemplo un nombre de una persona) debemos definir una variable de tipo `string`.

En realidad hemos estado utilizando en todos los problemas planteados desde el principio la definición de una variable de tipo `string` donde almacenamos cualquier dato que carga el operador por teclado, esto debido a que la clase `Console` tiene el método `ReadLine` que carga un `string`.

Más adelante veremos en profundidad y detenimiento los conceptos del manejo de `string`, por ahora solo nos interesa la mecánica para trabajar con cadenas de caracteres.

Problema I:

Solicitar el ingreso del nombre y edad de dos personas. Mostrar el nombre de la persona con mayor edad.

Programa:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace CadenaDeCaracteres1
{
    class Program
    {
        static void Main(string[] args)
        {
            String nombre1,nombre2;
            int edad1,edad2;
            String linea;
            Console.Write("Ingrese el nombre:");
```

```
nombre1=Console.ReadLine();
Console.Write("Ingrese edad:");
linea=Console.ReadLine();
edad1=int.Parse(linea);
Console.Write("Ingrese el nombre:");
nombre2=Console.ReadLine();
Console.Write("Ingrese edad:");
linea=Console.ReadLine();
edad2=int.Parse(linea);
Console.Write("La persona de mayor
edad es:");
if (edad1>edad2)
{
    Console.Write(nombre1);
}
else
{
    Console.Write(nombre2);
}
Console.ReadKey();
}
```

Para almacenar un nombre debemos definir una variable de tipo string y su ingreso por teclado se hace llamando al método ReadLine del objeto Console:

```
nombre1=Console.ReadLine();
```

No tenemos que hacer ninguna conversión como sucede cuando cargamos un valor de tipo int o float.

RECORDEMOS USO DE LA CADENA DE CARACTERES

Actividad:

- I. Confecciona el siguiente programa, solicitar el ingreso de dos apellidos. Mostrar un mensaje si son iguales o distintos.

LO QUE SABES DE DECLARACIÓN DE VARIABLES

Declaración de variables

La declaración de una variable simple, la forma más sencilla de un declarador directo, especifica el nombre y el tipo de la variable. También especifica la clase de almacenamiento y el tipo de datos de la variable. En las declaraciones de variables se requieren clases o tipos de almacenamiento (o ambos)

PARA EMPEZAR CON LA DECLARACIÓN DE UNA CLASE Y DEFINICIÓN DE OBJETOS

Declaración de una clase y definición de objetos

La programación orientada a objetos se basa en la programación de clases; a diferencia de la programación estructurada, que está centrada en las funciones.

Una clase es un molde del que luego se pueden crear múltiples objetos, con similares características.

Una clase es una plantilla (molde), que define atributos (variables) y métodos (funciones)

La clase define los atributos y métodos comunes a los objetos de ese tipo, pero luego, cada objeto tendrá sus propios valores y compartirán las mismas funciones.

Debemos crear una clase antes de poder crear objetos (instancias) de esa clase. Al crear un objeto de una clase, se dice que se crea una instancia de la clase o un objeto propiamente dicho.

La estructura de una clase es:

```
class [nombre de la clase] {  
    [atributos o variables de la clase]  
    [métodos o funciones de la clase]  
    [main]  
}
```

```
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
using System.Threading.Tasks;  
  
namespace PruebaClase1
```

```
{  
    class Persona  
    {  
        private string nombre;  
        private int edad;  
  
        public void Inicializar()  
        {  
            Console.Write("Ingrese el nombre:");  
            nombre = Console.ReadLine();  
            string linea;  
            Console.Write("Ingrese la edad:");  
            linea = Console.ReadLine();  
            edad = int.Parse(linea);  
        }  
  
        public void Imprimir()  
        {  
            Console.Write("Nombre:");  
            Console.WriteLine(nombre);  
            Console.Write("Edad:");  
            Console.WriteLine(edad);  
        }  
  
        public void EsMayorEdad()  
        {  
            if (edad >= 18)  
            {  
                Console.Write("Es mayor de edad");  
            }  
            else  
            {  
                Console.Write("No es mayor de  
edad");  
            }  
            Console.ReadKey();  
        }  
    }  
}
```

```
static void Main(string[] args)
{
    Persona perl = new Persona();
    perl.Inicializar();
    perl.Imprimir();
    perl.EsMayorEdad();
}
}
```

El nombre de la clase debe hacer referencia al concepto (en este caso la hemos llamado Persona):

```
class Persona
```

Los atributos los definimos dentro de la clase pero fuera de la main:

```
private string nombre;
private int edad;
```

Veremos más adelante que un atributo es normalmente definido con la cláusula `private` (con esto no permitimos el acceso al atributo desde otras clases)

A los atributos se tiene acceso desde cualquier función o método de la clase (salvo la main)

Luego de definir los atributos de la clase debemos declarar los métodos o funciones de la clase. La sintaxis es parecida a la main (sin la cláusula `static`):

```
public void Inicializar()
{
    Console.Write("Ingrese el nombre:");
    nombre = Console.ReadLine();
    string linea;
    Console.Write("Ingrese la edad:");
    linea = Console.ReadLine();
    edad = int.Parse(linea);
}
```

En el método inicializar (que será el primero que deberemos llamar desde la main) cargamos por teclado los atributos nombre y edad. Como podemos ver el método inicializar puede hacer acceso a dos atributos de la clase Persona. El segundo método tiene por objetivo imprimir el contenido de los atributos nombre y edad (los datos de los atributos se cargaron al ejecutarse previamente el método inicializar:

```
Console.Write("Nombre:");
```

```
Console.WriteLine(nombre);  
Console.Write("Edad:");  
Console.WriteLine(edad);
```

El tercer método tiene por objetivo mostrar un mensaje si la persona es mayor o no de edad:

```
public void EsMayorEdad()  
{  
    if (edad >= 18)  
    {  
        Console.Write("Es mayor de edad");  
    }  
    else  
    {  
        Console.Write("No es mayor de edad");  
    }  
    Console.ReadKey();  
}
```

Por último en la main declaramos un objeto de la clase Persona y llamamos a los métodos en un orden adecuado:

```
Persona perl = new Persona();  
perl.Inicializar();  
perl.Imprimir();  
perl.EsMayorEdad();
```

Persona perl = new Persona(); //Declaración y creación del objeto
perl.Inicializar(); //Llamada de un método

RECORDEMOS EL USO DE DECLARACIÓN DE UNA CLASE Y DEFINICIÓN DE OBJETOS

Actividad:

1. Desarrollar un programa que cargue los lados de un triángulo e implemente los siguientes métodos: inicializar los atributos, imprimir el valor del lado mayor y otro método que muestre si es equilátero o no.

MANOS A LA OBRA CON SEGUNDO PRODUCTO SEGUNDO PARCIAL

Nombre de la práctica: Modulo de programas

Nomenclatura: MI-SII Practica/modulo de programas

Temas: Cadena de caracteres y declaración de una clase y definición de objetos

Duración: 3 Sesiones

Objetivos: Emplear las cadenas de caracteres y declaraciones de clases con sus respectivas funciones, para la confección de programas con estructuras de control en C#

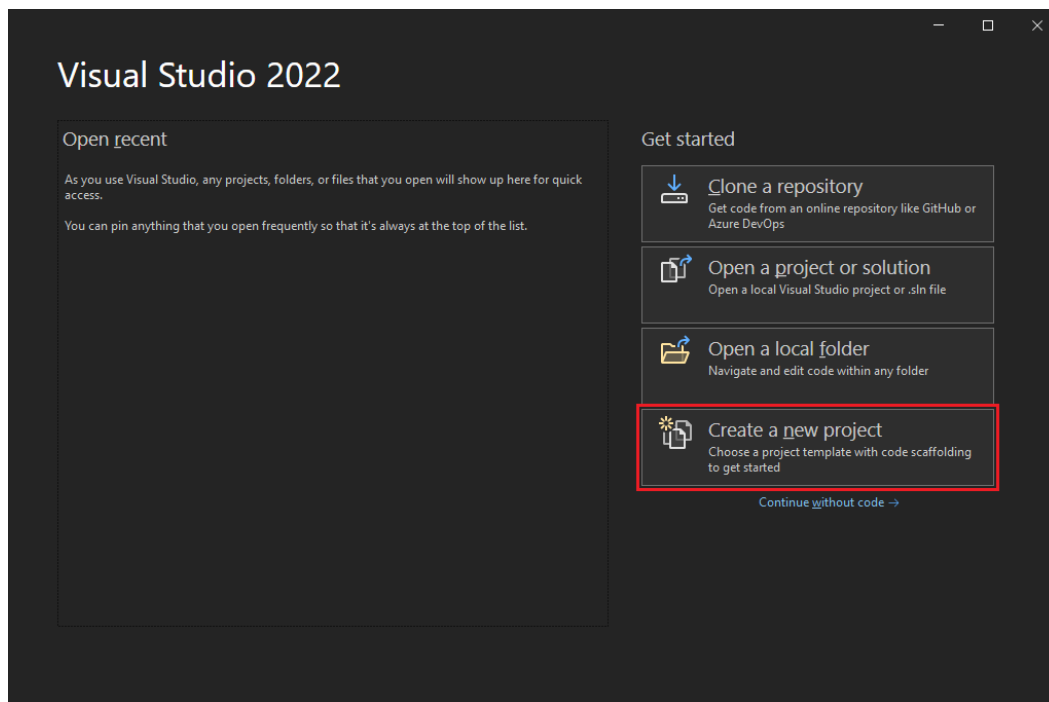
Materiales y equipo: Equipo de cómputo con software instalado Visual Studio o en su caso aplicación de Play Store

Procedimiento:

Crear un proyecto

2. Para empezar, cree un proyecto de aplicación de C#. En el tipo de proyecto se incluyen todos los archivos de plantilla que necesita.

Abra Visual Studio y elija **Crear un nuevo proyecto** en la ventana Inicio.



3. En la ventana **Crear un nuevo proyecto**, seleccione **Todos los lenguajes** y, a continuación, elija **C#** en la lista desplegable. Seleccione **Windows** en la lista **Todas las plataformas** y **Consola** en la lista **Todos los tipos de proyecto**.

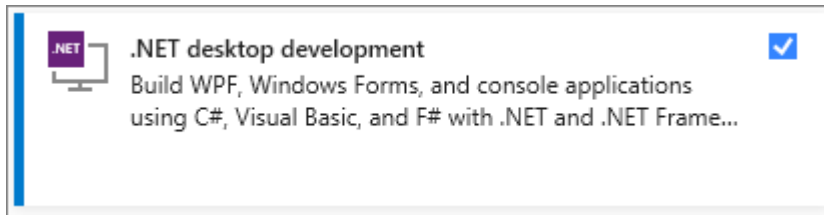
Después de aplicar los filtros de lenguaje, plataforma y tipo de proyecto, elija la plantilla **Aplicación de consola** y, luego, seleccione **Siguiente**.

Nota

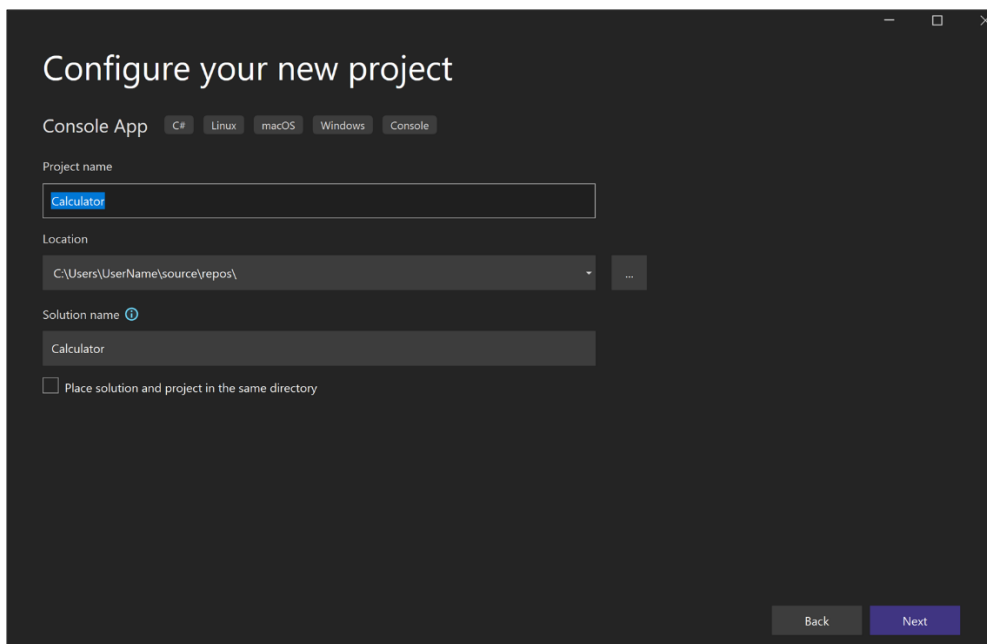
Si no ve la plantilla **Aplicación de consola**, seleccione **Instalar más herramientas y características**.

Not finding what you're looking for?
Install more tools and features

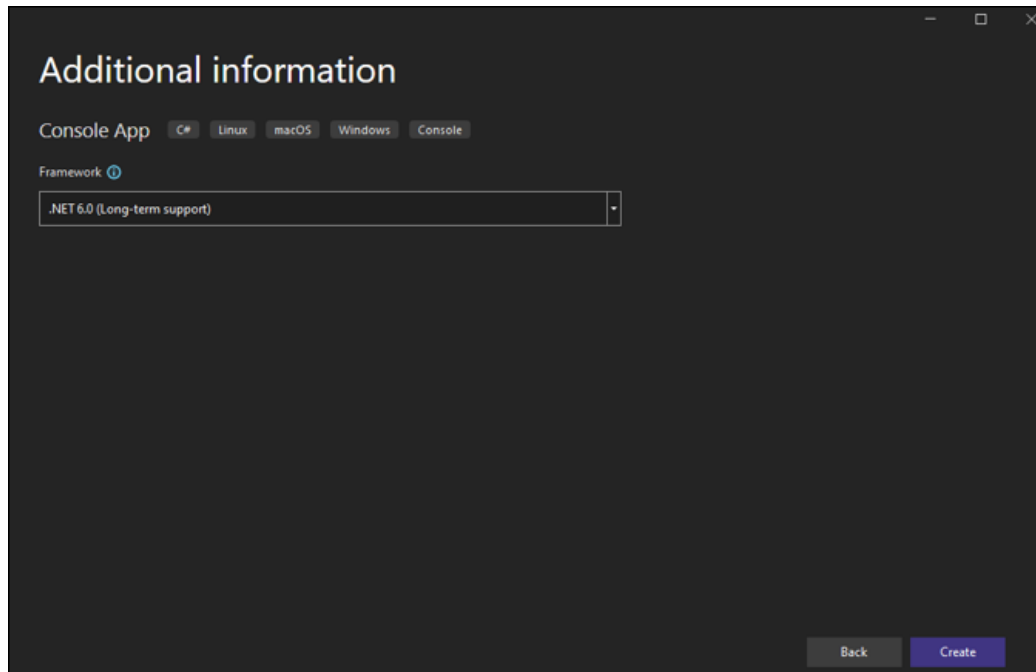
En el Instalador de Visual Studio, elija la carga de trabajo de **desarrollo de escritorio de .NET** y, a continuación, seleccione **Modificar**.



4. En la ventana **Configurar el nuevo proyecto**, escriba CALCULATOR en el cuadro **Nombre del proyecto** y, a continuación, seleccione **Siguiente**.



5. En la ventana **Información adicional**, **.NET 6.0** ya debe estar seleccionado para la plataforma de destino. Seleccione **Crear**.



Visual Studio abre el nuevo proyecto, que incluye código predeterminado de "Hola mundo".

Para verlo en el editor, seleccione el archivo de código Program.cs en la ventana del Explorador de soluciones, que normalmente se encuentra en el lado derecho de Visual Studio.

La instrucción de código única llama al método WriteLine para mostrar la cadena literal "¡Hola mundo!" en la ventana de consola. Si presiona F5, puede ejecutar el programa predeterminado en modo de depuración. Una vez que la aplicación se ejecuta en el depurador, la ventana de la consola permanece abierta. Presione cualquier tecla para cerrar la ventana de consola.

Nota

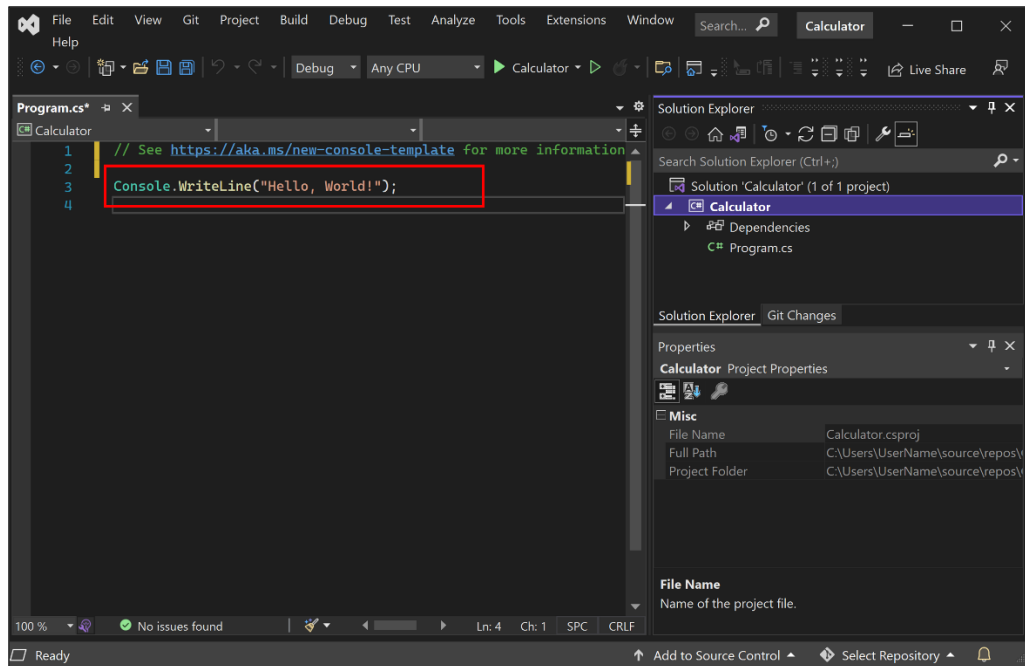
A partir de .NET 6, los nuevos proyectos que usan la plantilla de consola generan código diferente al de versiones anteriores. Para obtener más información, consulte la página Las nuevas plantillas de C# generan instrucciones de nivel superior.

Creación de la aplicación

Empiece con algunos cálculos básicos de enteros en C#.

6. En el **Explorador de soluciones**, en el panel derecho, seleccione **Program.cs** para mostrar el archivo en el editor de código

- En el editor de código, reemplace el código predeterminado "Hello World" que dice `Console.WriteLine("Hello World!");`.



- Ejecuta los siguientes bloques de programas

Programas a ejecutar :

- Confecciona el siguiente programa, solicitar el ingreso de dos apellidos. Mostrar un mensaje si son iguales o distintos.
- Desarrollar un programa que cargue los lados de un triángulo e implemente los siguientes métodos: inicializar los atributos, imprimir el valor del lado mayor y otro método que muestre si es equilátero o no.



INSTRUMENTO DE EVALUACIÓN DE SEGUNDA PRACTICA PARCIAL 2

Aspectos a evaluar	Ponderación		
	Porcentaje	sí	No
Portada con los datos de identificación	2		
Código del programa de cadena de caracteres	4		
Resultado del programa de cadena de caracteres	4		
El programa da el resultado de la diferencia de edades y con sus respectivos nombres	4		
Resultado del programa donde se declara una clase y sus objetos	4		
Código del programa donde se declara una clase y sus objetos	4		
El programa da los tipos de triángulos ingresados	4		
Tiene limpieza y formalidad	2		
Se entrega con un folder en condiciones optimas de un trabajo de media superior	2		
Nombre y firma del docente	Nombre y firma del padre tutor		

LO QUE SABES DE UN MÉTODO

Un método

Se define como un conjunto de instrucciones englobadas dentro de una sola instrucción, la cual se manda llamar cuando es necesario realizar una tarea. Cada programa en C# tiene un método principal, el cual se llama main.

PARA EMPEZAR DECLARACIÓN DE MÉTODOS

Cuando uno plantea una clase en lugar de especificar todo el algoritmo en un único método (lo que hicimos en los primeros pasos de este tutorial) es dividir todas las responsabilidades de la clase en un conjunto de métodos.

Un método hemos visto que tiene la siguiente sintaxis:

```
public void [nombre del método]()  
{  
    [algoritmo]  
}
```

Veremos que hay varios tipos de métodos:

Métodos con parámetros.

Un método puede tener parámetros:

```
public void [nombre del método]([parámetros])  
{  
    [algoritmo]  
}
```

Los parámetros los podemos imaginar como variables locales al método, pero su valor se inicializa con datos que llegan cuando lo llamamos.

Problema I:

Confeccionar una clase que permita ingresar valores enteros por teclado y nos muestre la tabla de multiplicar de dicho valor. Finalizar el programa al ingresar el -1.

Programa:



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Tabla
{
    class TablaMultiplicar
    {
        public void CargarValor()
        {
            int valor;
            string linea;
            do
            {
                Console.Write("Ingrese un valor (-
1 para finalizar):");
                linea = Console.ReadLine();
                valor = int.Parse(linea);
                if (valor != -1)
                {
                    Calcular(valor);
                }
            } while (valor != -1);
        }

        public void Calcular(int v)
        {
            for(int f=v;f<=v*10;f=f+v)
            {
                Console.Write(f+"-");
            }
            Console.WriteLine();
        }

        static void Main(string[] args)
        {

```

```
        TablaMultiplicar tm = new  
        TablaMultiplicar();  
        tm.CargarValor();  
    }  
}  
}
```

En esta clase no hemos definido ningún atributo.

El método `Calcular` recibe un parámetro de tipo entero, luego lo utilizamos dentro del método para mostrar la tabla de multiplicar de dicho valor, para esto inicializamos la variable `f` con el valor que llega en el parámetro. Luego de cada ejecución del `for` incrementamos el contador `f` con el valor de `v`.

```
public void Calcular(int v)  
{  
    for(int f=v;f<=v*10;f=f+v)  
    {  
        Console.Write(f+"-");  
    }  
    Console.WriteLine();  
}
```

Un método puede no tener parámetros como hemos visto en problemas anteriores o puede tener uno o más parámetros (en caso de tener más de un parámetro los mismos se separan por coma)

El método `CargarValores` no tiene parámetros y tiene por objetivo cargar un valor entero por teclado y llamar al método `Calcular` para que muestre la tabla de multiplicar del valor que le pasamos por teclado:

```
public void CargarValor()  
{  
    int valor;  
    string linea;  
    do  
    {  
        Console.Write("Ingrese un valor (-1 para  
finalizar):");  
        linea = Console.ReadLine();  
        valor = int.Parse(linea);  
        if (valor != -1)  
        {  
            Calcular(valor);  
        }  
    }  
}
```



```
        } while (valor != -1);  
    }
```

Como vemos al método Calcular lo llamamos por su nombre y entre paréntesis le pasamos el dato a enviar (debe ser un valor o variable entera)

En este problema en la Main solo llamamos al método CargarValor, ya que el método Calcular luego es llamado por el método CargarValor:

```
static void Main(string[] args)  
{  
    TablaMultiplicar tm = new TablaMultiplicar();  
    tm.CargarValor();  
}
```

Métodos que retornan un dato.

Un método puede retornar un dato:

```
public [tipo de dato] [nombre del método]([parámetros])  
{  
    [algoritmo]  
    return [tipo de dato]  
}
```

Cuando un método retorna un dato en vez de indicar la palabra clave void previo al nombre del método indicamos el tipo de dato que retorna. Luego dentro del algoritmo en el momento que queremos que finalice el mismo y retorne el dato empleamos la palabra clave return con el valor respectivo.

RECORDEMOS USO DE LOS MÉTODOS

Actividad:

1. **Confeccionar una clase que permita ingresar tres valores por teclado. Luego mostrar el mayor y el menor.**

MANOS A LA OBRA CON TERCER PRODUCTO SEGUNDO PARCIAL

Nombre de la práctica: Modulo de programas

Nomenclatura: MI-SII Practica/modulo de programas

Temas: Cadena de caracteres y declaración de una clase y definición de objetos

Duración: 3 Sesiones

Objetivos: *Emplear las cadenas de caracteres y declaraciones de clases con sus respectivas funciones, para la confección de programas con estructuras de control en C#*

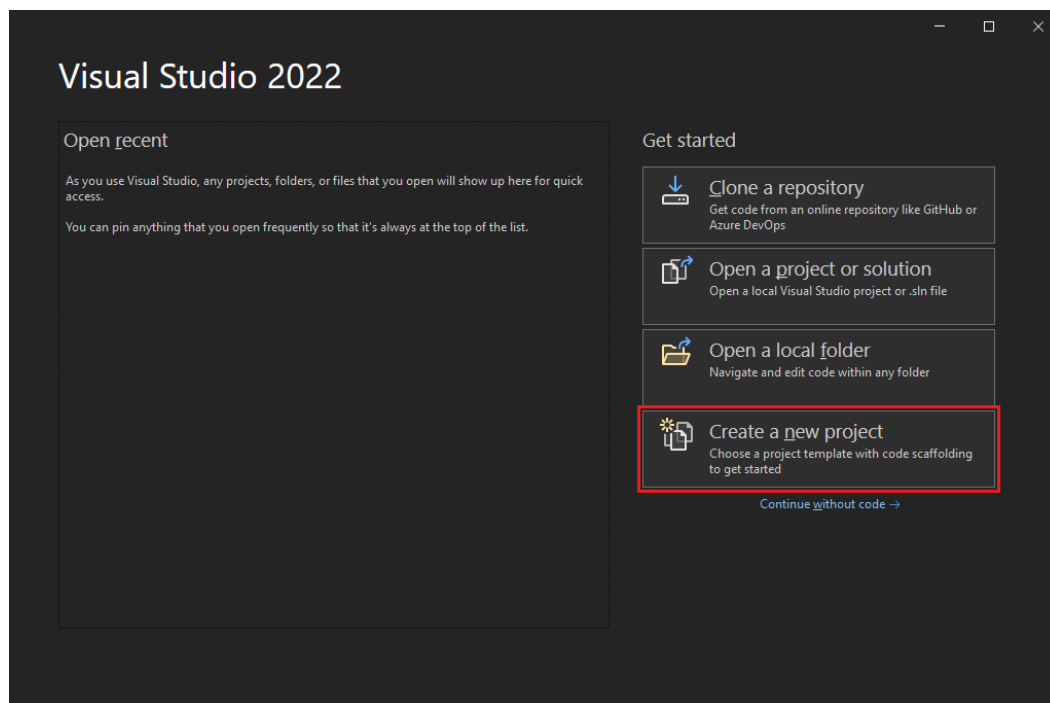
Materiales y equipo: Equipo de cómputo con software instalado Visual Studio o en su caso aplicación de Play Store

Procedimiento:

Crear un proyecto

1. Para empezar, cree un proyecto de aplicación de C#. En el tipo de proyecto se incluyen todos los archivos de plantilla que necesita.

Abra Visual Studio y elija **Crear un nuevo proyecto** en la ventana Inicio.



2. En la ventana **Crear un nuevo proyecto**, seleccione **Todos los lenguajes** y, a continuación, elija **C#** en la lista desplegable. Seleccione **Windows** en la lista **Todas las plataformas** y **Consola** en la lista **Todos los tipos de proyecto**.

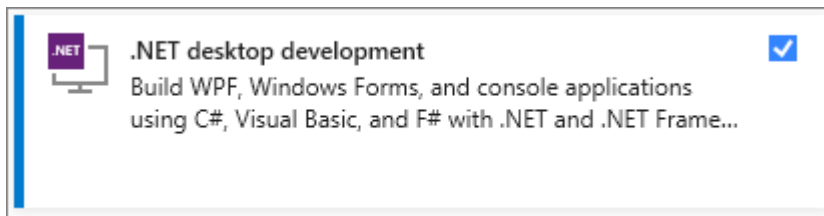
Después de aplicar los filtros de lenguaje, plataforma y tipo de proyecto, elija la plantilla **Aplicación de consola** y, luego, seleccione **Siguiente**.

Nota

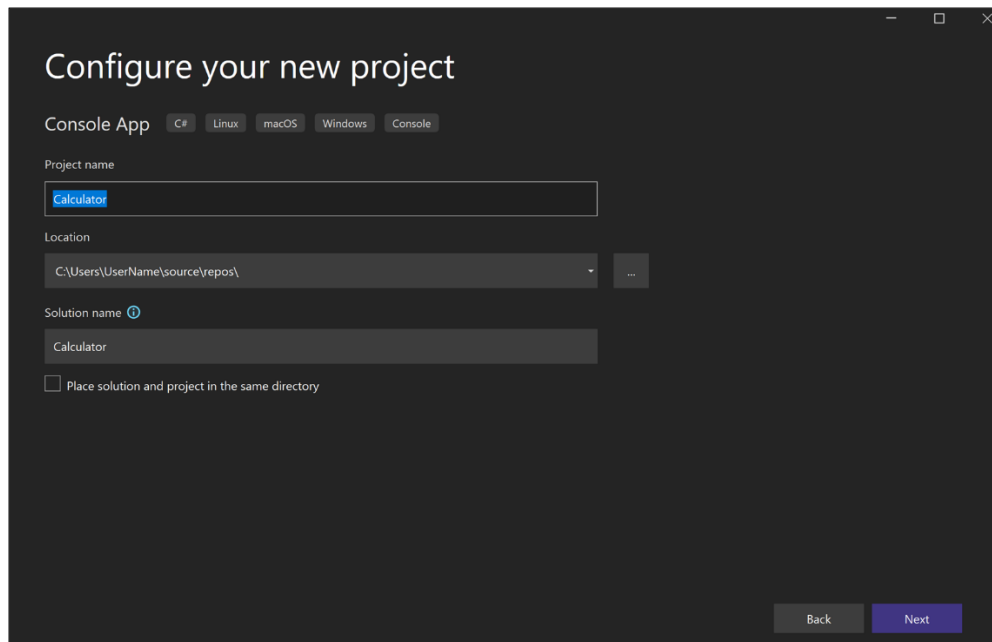
Si no ve la plantilla **Aplicación de consola**, seleccione **Instalar más herramientas y características**.

Not finding what you're looking for?
[Install more tools and features](#)

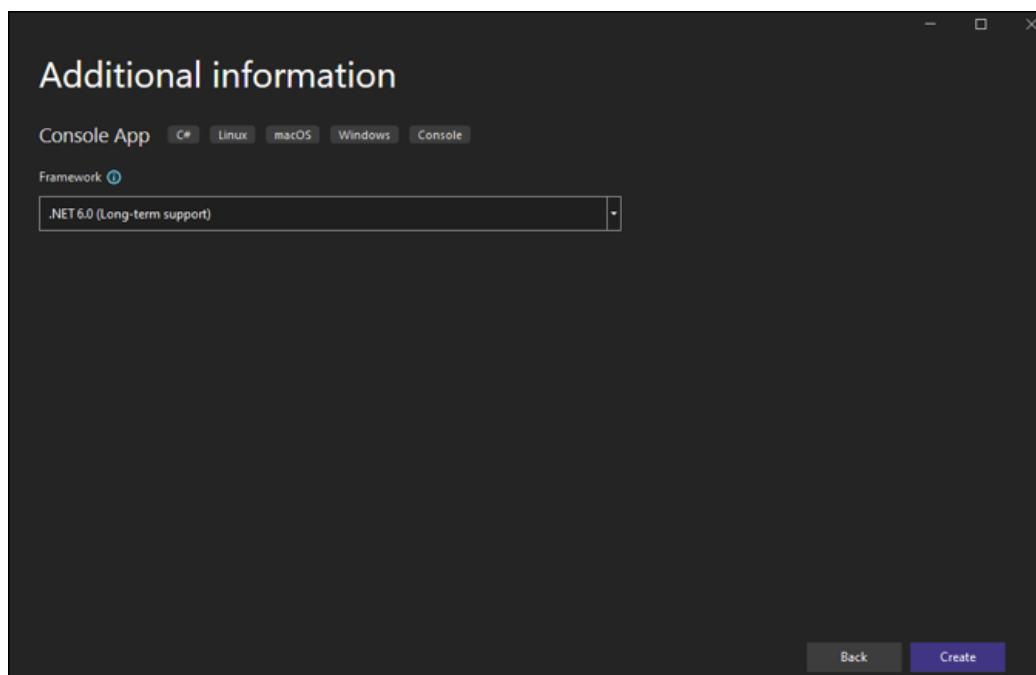
En el Instalador de Visual Studio, elija la carga de trabajo de **desarrollo de escritorio de .NET** y, a continuación, seleccione **Modificar**.



3. En la ventana **Configurar el nuevo proyecto**, escriba CALCULATOR en el cuadro **Nombre del proyecto** y, a continuación, seleccione **Siguiente**.



4. En la ventana **Información adicional**, **.NET 6.0** ya debe estar seleccionado para la plataforma de destino. Seleccione **Crear**.



Visual Studio abre el nuevo proyecto, que incluye código predeterminado de "Hola mundo".

Para verlo en el editor, seleccione el archivo de código Program.cs en la ventana del Explorador de soluciones, que normalmente se encuentra en el lado derecho de Visual Studio.

La instrucción de código única llama al método WriteLine para mostrar la cadena literal "¡Hola mundo!" en la ventana de consola. Si presiona F5, puede ejecutar el programa predeterminado en modo de depuración. Una vez que la aplicación se ejecuta en el depurador, la ventana de la consola permanece abierta. Presione cualquier tecla para cerrar la ventana de consola.

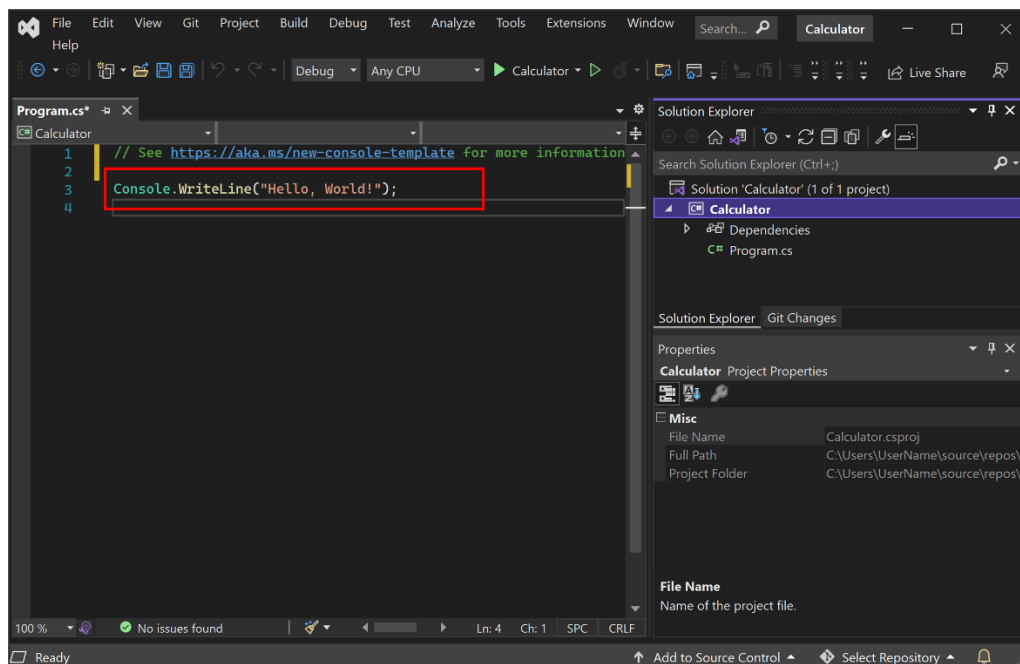
Nota

A partir de .NET 6, los nuevos proyectos que usan la plantilla de consola generan código diferente al de versiones anteriores. Para obtener más información, consulte la página Las nuevas plantillas de C# generan instrucciones de nivel superior.

Creación de la aplicación

Empiece con algunos cálculos básicos de enteros en C#.

5. En el **Explorador de soluciones**, en el panel derecho, seleccione **Program.cs** para mostrar el archivo en el editor de código
6. En el editor de código, reemplace el código predeterminado "Hello World" que dice `Console.WriteLine("Hello World!");`.



7. Ejecuta los siguientes bloques de programas

Confeccionar una clase que permita ingresar tres valores por teclado. Luego mostrar el mayor y el menor.

..... INSTRUMENTO DE EVALUACIÓN DE TERCER PRODUCTO PRACTICA PARCIAL 2

Aspectos a evaluar	Ponderación		
	Porcentaje	sí	No
Portada con los datos de identificación	2		
La impresión de pantalla es clara y fácil de interpretar	4		
Contiene el código visible y de fácil interpretación	5		
El programa muestra los métodos utilizados con claridad	5		
Tiene limpieza y formalidad	2		
Se entrega con un folder en condiciones óptimas de un trabajo de media superior	2		
Nombre y firma del docente	Nombre y firma del padre tutor		



MOMENTO DE ACTIVIDAD I

MANOS A LA OBRA

Nombre de la práctica:

Nomenclatura:

Tema:

Duración:

Objetivos: *(general y específico)*

Materiales y equipo:

Procedimiento:

INSTRUMENTO DE EVALUACIÓN

A LO QUE LLEGAMOS

Plantear interrogaciones que den muestra de lo aprendido en la práctica (reporte de práctica)



CIERRE

Aprendizaje clave:

PRÁCTICA AUTÓNOMA

Nombre de la práctica:

Nomenclatura:

Tema:

Duración:

Objetivos: *(general y específico)*

Materiales y equipo:

Procedimiento:

INSTRUMENTO DE EVALUACIÓN

A LO QUE LLEGAMOS

Plantear interrogaciones que den muestra de lo aprendido en la práctica (reporte de práctica)

EN RESUMEN

PROYECTO TRANSVERSAL



INSTRUMENTO DE EVALUACIÓN

GLOSARIO

RECURSOS DE APOYO



TERCER PARCIAL

ENCUADRE

COMPETENCIA PROFESIONAL

Reconoce las características de un lenguaje de programación estructurado

Desarrolla código en un lenguaje de programación estructurado

SITUACIÓN DE APRENDIZAJE



PLAN DE EVALUACIÓN

PRODUCTOS DE APRENDIZAJE TERCER PARCIAL

PRODUCTO	PONDERACIÓN	PORCENTAJE OBTENIDO	OBSERVACIONES
Confección de programa con ciclo for	30%		
Carrusel de programas	30%		
Programa juego de opciones	20%		
Tutorías y Orientación Educativa	10%		
Innovación y liderazgo CECyTE	10%		

INICIO

Aprendizajes claves

LO QUE SABES DE

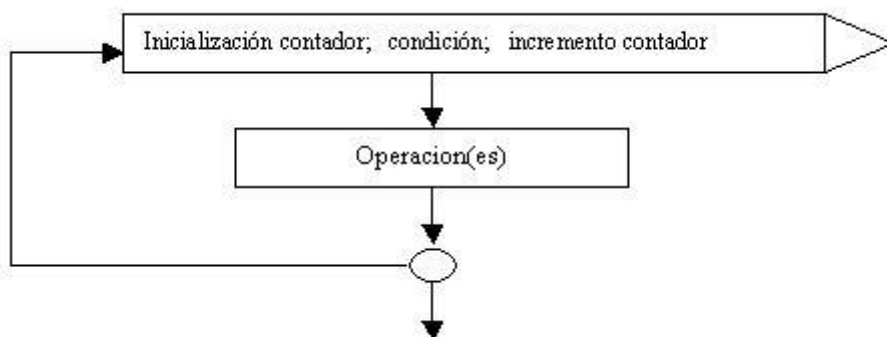


Un FOR en programación se usa cuando queremos repetir un conjunto de instrucciones un número finito de veces

PARA EMPEZAR CICLO FOR

Cualquier problema que requiera una estructura repetitiva se puede resolver empleando la estructura while. Pero hay otra estructura repetitiva cuyo planteo es más sencillo en ciertas situaciones. En general, la estructura for se usa en aquellas situaciones en las cuales **CONOCEMOS** la cantidad de veces que queremos que se ejecute el bloque de instrucciones. Ejemplo: cargar 10 números, ingresar 5 notas de alumnos, etc. Conocemos de antemano la cantidad de veces que queremos que el bloque se repita. Veremos, sin embargo, que en el lenguaje C# la estructura for puede usarse en cualquier situación repetitiva, porque en última instancia no es otra cosa que una estructura while generalizada.

Representación gráfica:

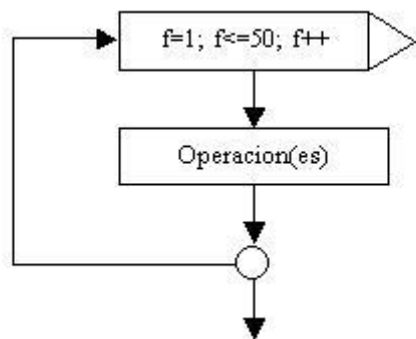


En su forma más típica y básica, esta estructura requiere una variable entera que cumple la función de un **CONTADOR** de vueltas. En la sección indicada como

"inicialización contador", se suele colocar el nombre de la variable que hará de contador, asignándole a dicha variable un valor inicial. En la sección de "condición" se coloca la condición que deberá ser verdadera para que el ciclo continúe (en caso de un falso, el ciclo se detendrá). Y finalmente, en la sección de "incremento contador" se coloca una instrucción que permite modificar el valor de la variable que hace de contador (para permitir que alguna vez la condición sea falsa)

Cuando el ciclo comienza, antes de dar la primera vuelta, la variable del for toma el valor indicado en la sección de "inicialización contador". Inmediatamente se verifica, en forma automática, si la condición es verdadera. En caso de serlo se ejecuta el bloque de operaciones del ciclo, y al finalizar el mismo se ejecuta la instrucción que se haya colocado en la tercer sección. Seguidamente, se vuelve a controlar el valor de la condición, y así prosigue hasta que dicha condición entregue un falso.

Si conocemos la cantidad de veces que se repite el bloque es muy sencillo emplear un for, por ejemplo si queremos que se repita 50 veces el bloque de instrucciones puede hacerse así:



La variable del for puede tener cualquier nombre. En este ejemplo se la ha definido con el nombre f.

Analicemos el ejemplo:

- La variable f toma inicialmente el valor 1.
- Se controla automáticamente el valor de la condición: como f vale 1 y esto es menor que 50, la condición da verdadero.
- Como la condición fue verdadera, se ejecutan la/s operación/es.
- Al finalizar de ejecutarlas, se retorna a la instrucción f++, por lo que la variable f se incrementa en uno.
- Se vuelve a controlar (automáticamente) si f es menor o igual a 50.

Como ahora su valor es 2, se ejecuta nuevamente el bloque de instrucciones e incrementa nuevamente la variable del for al terminar el mismo.

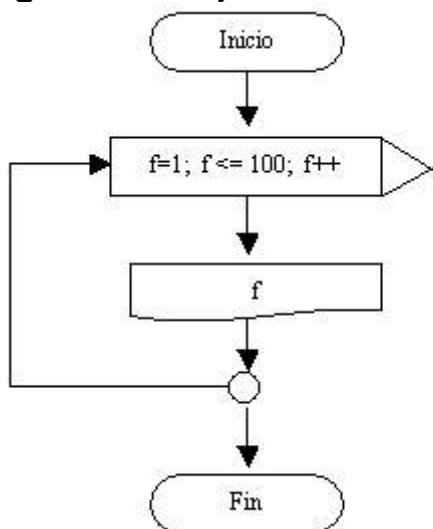
- El proceso se repetirá hasta que la variable f sea incrementada al valor 51.

En este momento la condición será falsa, y el ciclo se detendrá. La variable *f* PUEDE ser modificada dentro del bloque de operaciones del *for*, aunque esto podría causar problemas de lógica si el programador es inexperto. La variable *f* puede ser inicializada en cualquier valor y finalizar en cualquier valor. Además, no es obligatorio que la instrucción de modificación sea un incremento del tipo contador (*f++*). Cualquier instrucción que modifique el valor de la variable es válida. Si por ejemplo se escribe *f=f+2* en lugar de *f++*, el valor de *f* será incrementado de a 2 en cada vuelta, y no de a 1. En este caso, esto significará que el ciclo no efectuará las 50 vueltas sino sólo 25.

Problema 1:

Realizar un programa que imprima en pantalla los números del 1 al 100.

Diagrama de flujo:



Podemos observar y comparar con el problema realizado con el *while*. Con la estructura *while* el *CONTADOR* *x* sirve para contar las vueltas. Con el *for* el *CONTADOR* *f* cumple dicha función. Inicialmente *f* vale 1 y como no es superior a 100 se ejecuta el bloque, imprimimos el contenido de *f*, al finalizar el bloque repetitivo se incrementa la variable *f* en 1, como 2 no es superior a 100 se repite el bloque de instrucciones. Cuando la variable del *for* llega a 101 sale de la estructura repetitiva y continúa la ejecución del algoritmo que se indica después del círculo. La variable *f* (o como sea que se decida llamarla) debe estar definida como una variable más.

Programa:

```
using System;
using System.Collections.Generic;
using System.Linq;
```



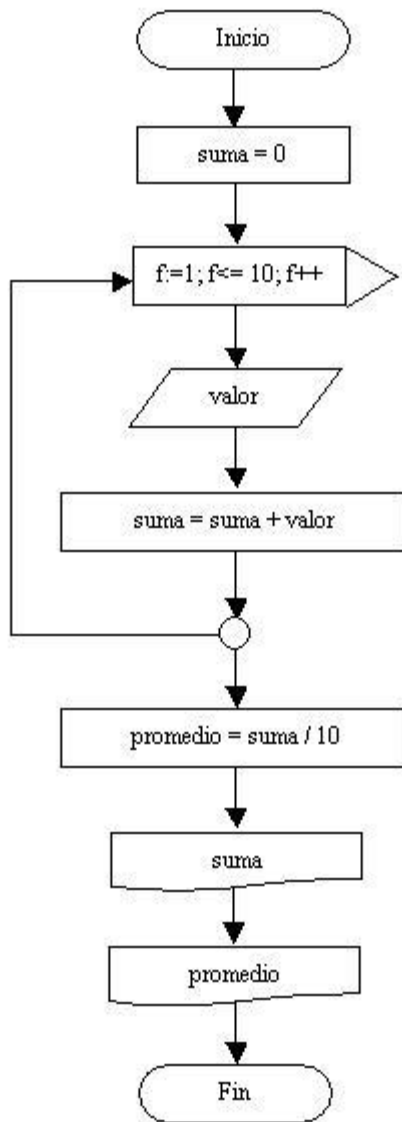
```
using System.Text;
using System.Threading.Tasks;

namespace EstructuraRepetitivaFor1
{
    class Program
    {
        static void Main(string[] args)
        {
            int f;
            for (f=1; f<=100; f++)
            {
                Console.Write(f);
                Console.Write("-");
            }
            Console.ReadKey();
        }
    }
}
```

Problema 2:

Desarrollar un programa que permita la carga de 10 valores por teclado y nos muestre posteriormente la suma de los valores ingresados y su promedio. Este problema ya lo desarrollaremos empleando la estructura for.

Diagrama de flujo:



En este caso, a la variable del for (f) sólo se la requiere para que se repita el bloque de instrucciones 10 veces.

Programa:

Ver video

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace EstructuraRepetitivaFor2
```



```
{
    class Program
    {
        static void Main(string[] args)
        {
            int suma,f,valor,promedio;
            string linea;
            suma=0;
            for(f=1;f<=10;f++)
            {
                Console.Write("Ingrese valor:");
                linea=Console.ReadLine();
                valor=int.Parse(linea);
                suma=suma+valor;
            }
            Console.Write("La suma es:");
            Console.WriteLine(suma);
            promedio=suma/10;
            Console.Write("El promedio es:");
            Console.Write(promedio);
            Console.ReadKey();
        }
    }
}
```

El problema requiere que se carguen 10 valores y se sumen los mismos. Tener en cuenta encerrar entre llaves bloque de instrucciones a repetir dentro del for.

El promedio se calcula fuera del for luego de haber cargado los 10 valores.

RECORDEMOS USO Y APLICACIÓN DEL CICLO FOR

Actividad:

1. Confeccionar un programa que lea n pares de datos, cada par de datos corresponde a la medida de la base y la altura de un triángulo. El programa deberá informar:
 - a) De cada triángulo la medida de su base, su altura y su superficie.
 - b) La cantidad de triángulos cuya superficie es mayor a 12.
2. Desarrollar un programa que solicite la carga de 10 números e imprima la suma de los últimos 5 valores ingresados.

3. Desarrollar un programa que muestre la tabla de multiplicar del 5 (del 5 al 50)

MANOS A LA OBRA CON PRIMER PRODUCTO TERCER PARCIAL

Nombre de la práctica: Programas matemáticos

Nomenclatura: MI-SII Practica/ Programas matemáticos

Temas: Ciclo FOR

Duración: 3 Sesiones

Objetivos: *Emplear los ciclos FOR para la confección de programas que ayuden a resolver problemas de índole matemáticas*

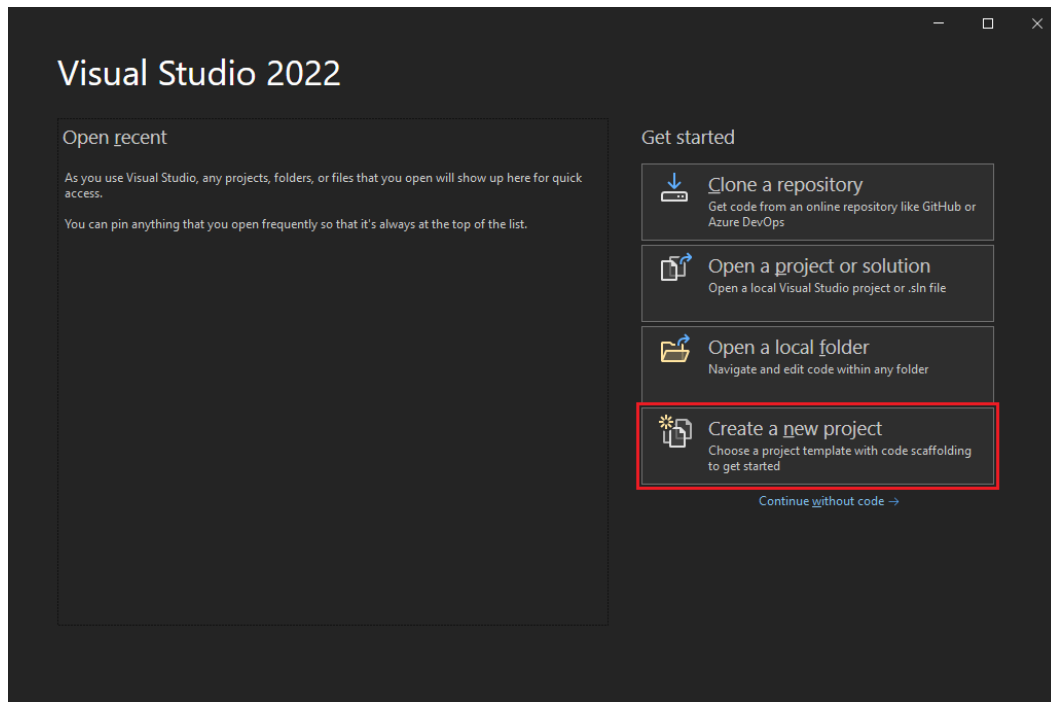
Materiales y equipo: Equipo de cómputo con software instalado Visual Studio o en su caso aplicación de Play Store

Procedimiento:

Crear un proyecto

1. Para empezar, cree un proyecto de aplicación de C#. En el tipo de proyecto se incluyen todos los archivos de plantilla que necesita.

Abra Visual Studio y elija **Crear un nuevo proyecto** en la ventana Inicio.



2. En la ventana **Crear un nuevo proyecto**, seleccione **Todos los lenguajes** y, a continuación, elija **C#** en la lista desplegable. Seleccione **Windows** en la lista **Todas las plataformas** y **Consola** en la lista **Todos los tipos de proyecto**.

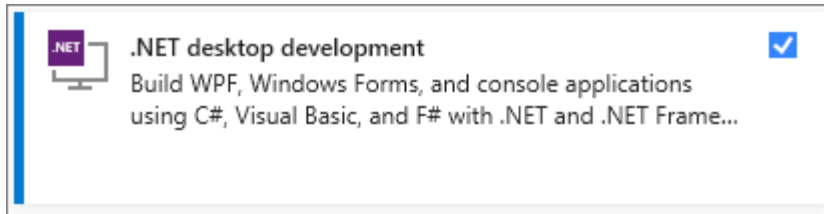
Después de aplicar los filtros de lenguaje, plataforma y tipo de proyecto, elija la plantilla **Aplicación de consola** y, luego, seleccione **Siguiente**.

Nota

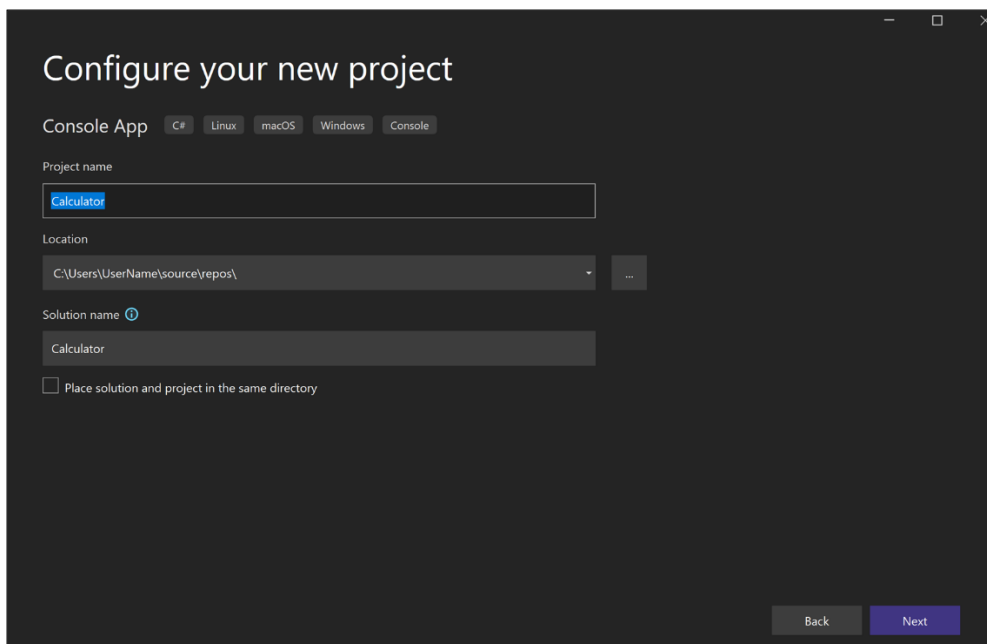
Si no ve la plantilla **Aplicación de consola**, seleccione **Instalar más herramientas y características**.

Not finding what you're looking for?
[Install more tools and features](#)

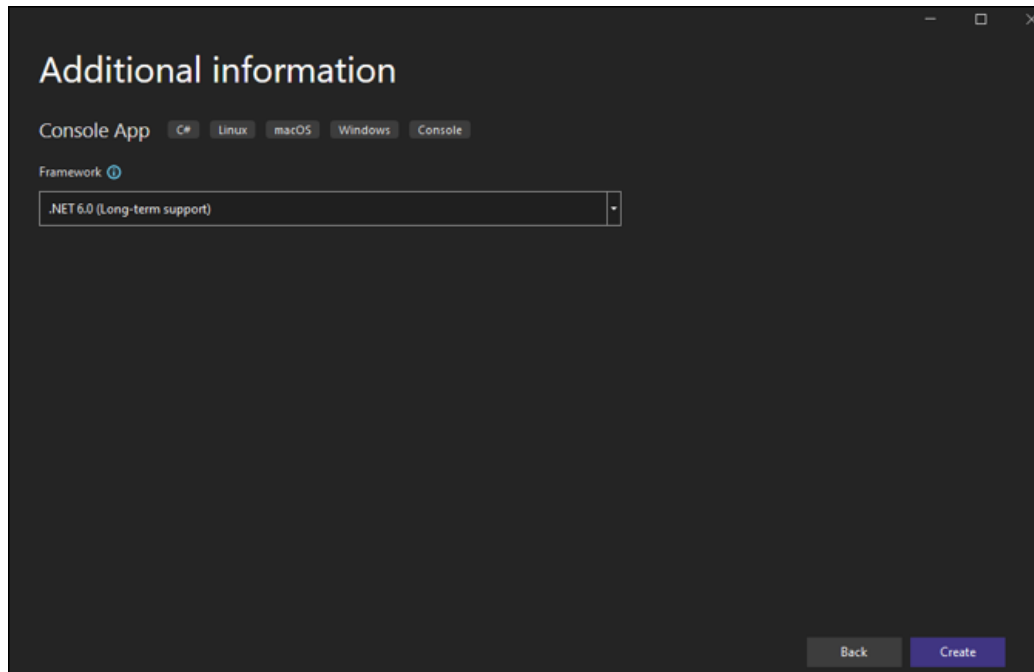
En el Instalador de Visual Studio, elija la carga de trabajo de **desarrollo de escritorio de .NET** y, a continuación, seleccione **Modificar**.



3. En la ventana **Configurar el nuevo proyecto**, escriba CALCULATOR en el cuadro **Nombre del proyecto** y, a continuación, seleccione **Siguiente**.



4. En la ventana **Información adicional**, **.NET 6.0** ya debe estar seleccionado para la plataforma de destino. Seleccione **Crear**.



Visual Studio abre el nuevo proyecto, que incluye código predeterminado de "Hola mundo".

Para verlo en el editor, seleccione el archivo de código Program.cs en la ventana del Explorador de soluciones, que normalmente se encuentra en el lado derecho de Visual Studio.

La instrucción de código única llama al método WriteLine para mostrar la cadena literal "¡Hola mundo!" en la ventana de consola. Si presiona F5, puede ejecutar el programa predeterminado en modo de depuración. Una vez que la aplicación se ejecuta en el depurador, la ventana de la consola permanece abierta. Presione cualquier tecla para cerrar la ventana de consola.

Nota

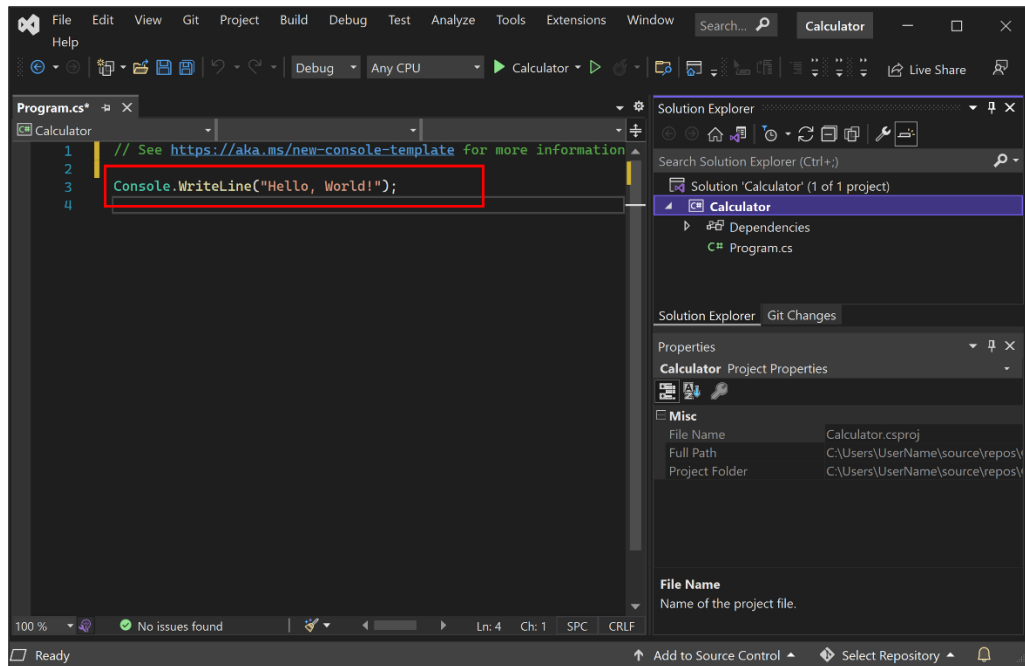
A partir de .NET 6, los nuevos proyectos que usan la plantilla de consola generan código diferente al de versiones anteriores. Para obtener más información, consulte la página Las nuevas plantillas de C# generan instrucciones de nivel superior.

Creación de la aplicación

Empiece con algunos cálculos básicos de enteros en C#.

5. En el **Explorador de soluciones**, en el panel derecho, seleccione **Program.cs** para mostrar el archivo en el editor de código

6. En el editor de código, reemplace el código predeterminado "Hello World" que dice `Console.WriteLine("Hello World!");`;



7. Ejecuta los siguientes bloques de programas

1. Confeccionar un programa que lea n pares de datos, cada par de datos corresponde a la medida de la base y la altura de un triángulo. El programa deberá informar:
 - a) De cada triángulo la medida de su base, su altura y su superficie.
 - b) La cantidad de triángulos cuya superficie es mayor a 12
2. Desarrollar un programa que solicite la carga de 10 números e imprima la suma de los últimos 5 valores ingresados.
3. Desarrollar un programa que muestre la tabla de multiplicar del 5 (del 5 al 50)

Aspectos a evaluar	Ponderación		
	Porcentaje	sí	No
Portada con los datos de identificación	2		
Código del programa que da lectura a n pares de datos de un triangulo	5		
Resultado del programa que da lectura a n pares de datos de un triangulo	5		
Código del programa que solicite la carga de 10 números e imprima la suma de los últimos 5 valores ingresados	5		
Resultado del programa que solicite la carga de 10 números e imprima la suma de los últimos 5 valores ingresados	5		
Código del programa que muestre la tabla de multiplicar del 5 (del 5 al 50	5		
Resultado del programa que muestre la tabla de multiplicar del 5 (del 5 al 50			
Tiene limpieza y formalidad	2		
Se entrega con un folder en condiciones optimas de un trabajo de media superior	1		
Nombre y firma del docente	Nombre y firma del padre tutor		



La estructura secuencial o programación secuencial se entiende como una metodología que basa su funcionamiento en tener acciones o instrucciones que sigan a otras de forma secuencial.

PARA EMPEZAR ESTRUCTURA DE PROGRAMACIÓN SECUENCIAL

Estructura secuencial de programación

Cuando en un problema sólo participan operaciones, entradas y salidas se la denomina una estructura secuencial. Los problemas diagramados y codificados previamente emplean solo estructuras secuenciales.

La programación requiere una práctica ininterrumpida de diagramación y codificación de problemas.

Problema:

Realizar la carga de dos números enteros por teclado e imprimir su suma y su producto.

Diagrama de flujo:

Tenemos dos entradas num1 y num2, dos operaciones: realización de la suma y del producto de los valores ingresados y dos salidas, que son los resultados de la suma y el producto de los valores ingresados. En el símbolo de impresión podemos indicar una o más salidas, eso queda a criterio del programador, lo mismo para indicar las entradas por teclado.

Programa:

```
using System;  
using System.Collections.Generic;
```



```
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SumaProductoNumeros
{
    class Program
    {
        static void Main(string[] args)
        {
            int num1, num2, suma, producto;
            string linea;
            Console.Write("Ingrese primer
valor:");
            linea = Console.ReadLine();
            num1 = int.Parse(linea);
            Console.Write("Ingrese segundo
valor:");
            linea = Console.ReadLine();
            num2 = int.Parse(linea);
            suma = num1 + num2;
            producto = num1 * num2;
            Console.Write("La suma de los dos
valores es:");
            Console.WriteLine(suma);
            Console.Write("El producto de los dos
valores es:");
            Console.WriteLine(producto);
            Console.ReadKey();
        }
    }
}
```

Recordemos que tenemos que seguir todos los pasos vistos para la creación de un proyecto.

Algunas cosas nuevas que podemos notar:

Podemos definir varias variables en la misma línea:

```
int num1, num2, suma, producto;
```

Si llamamos a la función `WriteLine` en lugar de `Write`, la impresión siguiente se efectuará en la próxima línea:

```
Console.WriteLine(suma);
```

RECORDEMOS USO Y APLICACIÓN DE ESTRUCTURA SECUENCIAL DE PROGRAMACIÓN

1. Realizar la carga del lado de un cuadrado, mostrar por pantalla el perímetro del mismo (El perímetro de un cuadrado se calcula multiplicando el valor del lado por cuatro)
2. Escribir un programa en el cual se ingresen cuatro números, calcular e informar la suma de los dos primeros y el producto del tercero y el cuarto.
3. Realizar un programa que lea cuatro valores numéricos e informar su suma y promedio.

LO QUE SABES DEL CILO WHILE

El bucle while o bucle mientras es un ciclo repetitivo basado en los resultados de una expresión lógica; se encuentra en la mayoría de los lenguajes de programación estructurados

PARA EMPEZAR CON ESTRUCTURA WHILE

Estructura WHILE

Hasta ahora hemos empleado estructuras SECUENCIALES y CONDICIONALES. Existe otro tipo de estructuras tan importantes como las anteriores que son las estructuras REPETITIVAS.

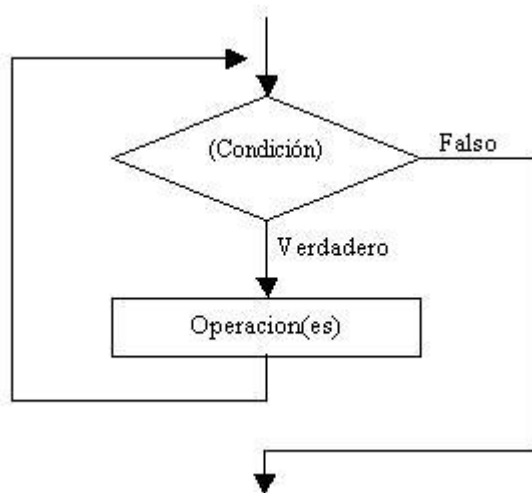
Una estructura repetitiva permite ejecutar una instrucción o un conjunto de instrucciones varias veces.

Una ejecución repetitiva de sentencias se caracteriza por:

- La o las sentencias que se repiten.
- El test o prueba de condición antes de cada repetición, que motivará que se repitan o no las sentencias.

Estructura repetitiva while.

Representación gráfica de la estructura while:



No debemos confundir la representación gráfica de la estructura repetitiva while (Mientras) con la estructura condicional if (Si)

Funcionamiento: En primer lugar se verifica la condición, si la misma resulta verdadera se ejecutan las operaciones que indicamos por la rama del Verdadero. A la rama del verdadero la graficamos en la parte inferior de la condición. Una línea al final del bloque de repetición la conecta con la parte superior de la estructura repetitiva.

En caso que la condición sea Falsa continúa por la rama del Falso y sale de la estructura repetitiva para continuar con la ejecución del algoritmo.

El bloque se repite MIENTRAS la condición sea Verdadera.

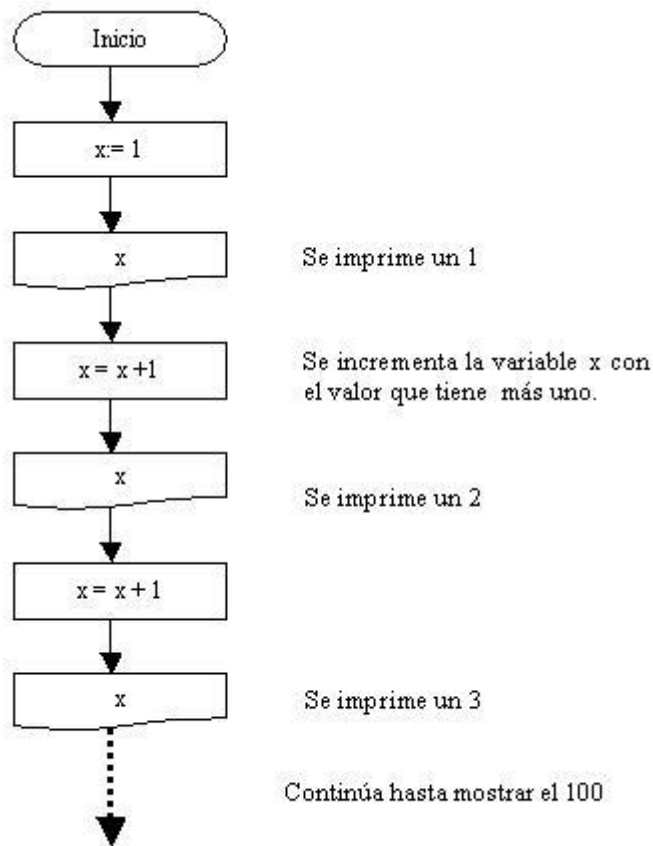
Importante: Si la condición siempre retorna verdadero estamos en presencia de un ciclo repetitivo infinito. Dicha situación es un error de programación, nunca finalizará el programa.

Problema 1:

Realizar un programa que imprima en pantalla los números del 1 al 100.

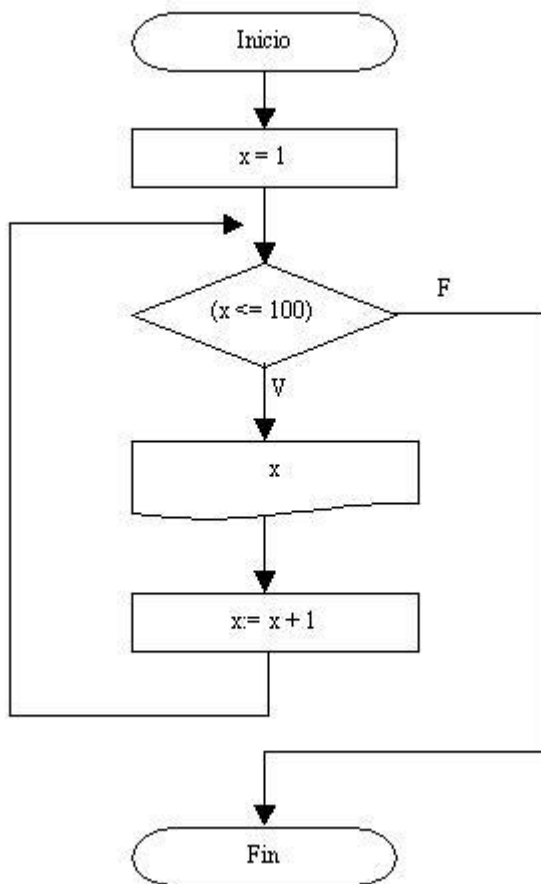
Sin conocer las estructuras repetitivas podemos resolver el problema empleando una estructura secuencial. Inicializamos una variable con el valor 1, luego imprimimos la variable, incrementamos nuevamente la variable y así sucesivamente.

Diagrama de flujo:



Si continuamos con el diagrama no nos alcanzarían las próximas 5 páginas para finalizarlo. Emplear una estructura secuencial para resolver este problema produce un diagrama de flujo y un programa en C# muy largo.

Ahora veamos la solución empleando una estructura repetitiva while:



Es muy importante analizar este diagrama: La primera operación inicializa la variable x en 1, seguidamente comienza la estructura repetitiva while y disponemos la siguiente condición ($x \leq 100$), se lee MIENTRAS la variable x sea menor o igual a 100.

Al ejecutarse la condición retorna VERDADERO porque el contenido de x (1) es menor o igual a 100. Al ser la condición verdadera se ejecuta el bloque de instrucciones que contiene la estructura while. El bloque de instrucciones contiene una salida y una operación. Se imprime el contenido de x , y seguidamente se incrementa la variable x en uno. La operación $x = x + 1$ se lee como "en la variable x se guarda el contenido de x más 1". Es decir, si x contiene 1 luego de ejecutarse esta operación se almacenará en x un 2.

Al finalizar el bloque de instrucciones que contiene la estructura repetitiva se verifica nuevamente la condición de la estructura repetitiva y se repite el proceso explicado anteriormente.

Mientras la condición retorne verdadero se ejecuta el bloque de instrucciones; al retornar falso la verificación de la condición se sale de la estructura repetitiva y continua el algoritmo, en este caso finaliza el programa.

Lo más difícil es la definición de la condición de la estructura while y qué bloque de instrucciones se van a repetir. Observar que si, por ejemplo, disponemos la condición $x \geq 100$ (si x es mayor o igual a 100) no provoca ningún error sintáctico pero estamos en presencia de un error lógico porque al evaluarse por primera vez la condición retorna falso y no se ejecuta el bloque de instrucciones que queríamos repetir 100 veces.

No existe una RECETA para definir una condición de una estructura repetitiva, sino que se logra con una práctica continua solucionando problemas.

Una vez planteado el diagrama debemos verificar si el mismo es una solución válida al problema (en este caso se debe imprimir los números del 1 al 100 en pantalla), para ello podemos hacer un seguimiento del flujo del diagrama y los valores que toman las variables a lo largo de la ejecución:

```
x
1
2
3
4
.
.
100
101 Cuando x vale 101 la condición de la estructura
repetitiva retorna falso,
en este caso finaliza el diagrama.
```

Importante: Podemos observar que el bloque repetitivo puede no ejecutarse ninguna vez si la condición retorna falso la primera vez. La variable x debe estar inicializada con algún valor antes que se ejecute la operación $x = x + 1$ en caso de no estar inicializada aparece un error de compilación.

Programa:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace EstructuraRepetitivaWhile1
{
    class Program
    {
        static void Main(string[] args)
        {
```

```
int x;  
x = 1;  
while (x <= 100)  
{  
    Console.Write(x);  
    Console.Write(" - ");  
    x = x + 1;  
}  
Console.ReadKey();  
}  
}
```

Recordemos que un problema no estará 100% solucionado si no hacemos el programa en C# que muestre los resultados buscados.

Problemos algunas modificaciones de este programa y veamos que cambios se deberían hacer para:

- 1 - Imprimir los números del 1 al 500.
- 2 - Imprimir los números del 50 al 100.
- 3 - Imprimir los números del -50 al 0.
- 4 - Imprimir los números del 2 al 100 pero de 2 en 2 (2,4,6,8100).

Respuestas:

- 1 - Debemos cambiar la condición del while con `x<=500`.
- 2 - Debemos inicializar x con el valor 50.
- 3 - Inicializar x con el valor -50 y fijar la condición `x<=0`.
- 4 - Inicializar a x con el valor 2 y dentro del bloque repetitivo incrementar a x en 2
(`x = x + 2`)

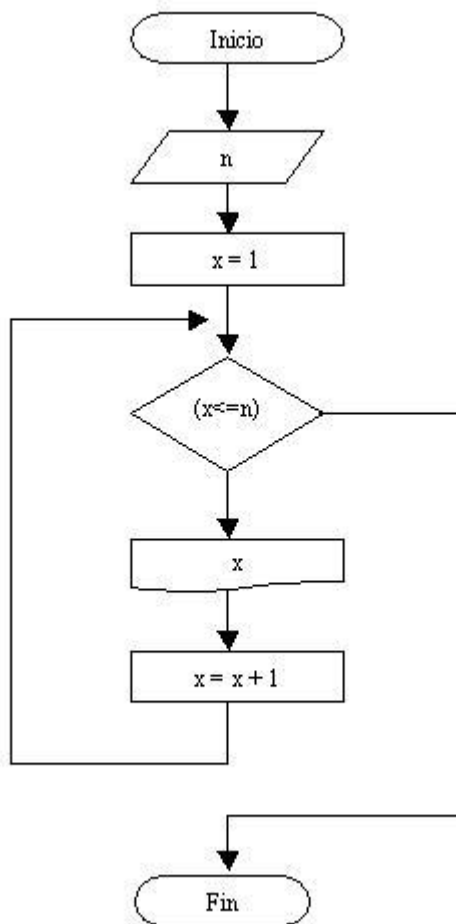
Problema 2:

Escribir un programa que solicite la carga de un valor positivo y nos muestre desde 1 hasta el valor ingresado de uno en uno.

Ejemplo: Si ingresamos 30 se debe mostrar en pantalla los números del 1 al 30.

Es de FUNDAMENTAL importancia analizar los diagramas de flujo y la posterior codificación en C# de los siguientes problemas, en varios problemas se presentan otras situaciones no vistas en el ejercicio anterior.

Diagrama de flujo:



Podemos observar que se ingresa por teclado la variable n . El operador puede cargar cualquier valor.

Si el operador carga 10 el bloque repetitivo se ejecutará 10 veces, ya que la condición es “Mientras $x \leq n$ ”, es decir “mientras x sea menor o igual a 10”; pues x comienza en uno y se incrementa en uno cada vez que se ejecuta el bloque repetitivo.

A la prueba del diagrama la podemos realizar dándole valores a las variables; por ejemplo, si ingresamos 5 el seguimiento es el siguiente:

n	x	
5	1	(Se imprime el contenido de x)
	2	" "
	3	" "
	4	" "
	5	" "
	6	(Sale del while porque 6 no es menor o igual a 5)

Programa:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace EstructuraRepetitivaWhile2
{
    class Program
    {
        static void Main(string[] args)
        {
            int n,x;
            string linea;
            Console.Write("Ingrese el valor
final:");
            linea=Console.ReadLine();
            n=int.Parse(linea);
            x=1;
            while (x<=n)
            {
                Console.Write(x);
                Console.Write(" - ");
                x = x + 1;
            }
            Console.ReadKey();
        }
    }
}
```

Los nombres de las variables **n** y **x** pueden ser palabras o letras (como en este caso)

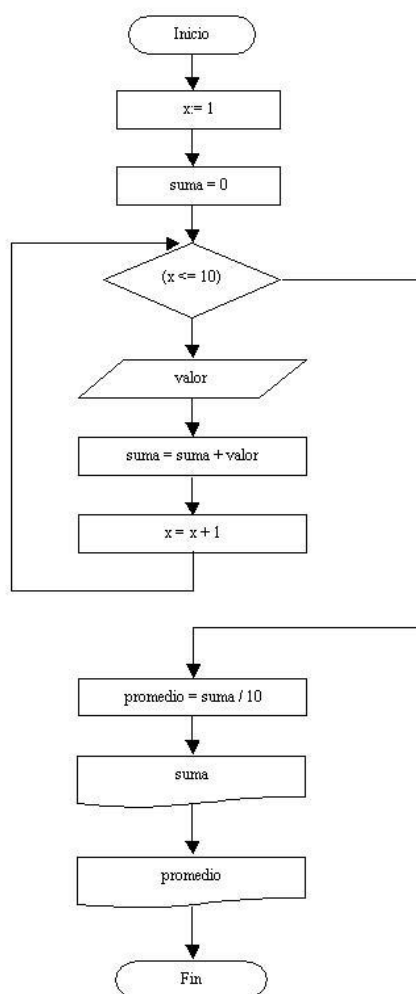
La variable **x** recibe el nombre de **CONTADOR**. Un contador es un tipo especial de variable que se incrementa o decrementa con valores constantes durante la ejecución del programa.

El contador x nos indica en cada momento la cantidad de valores impresos en pantalla.

Problema 3:

Desarrollar un programa que permita la carga de 10 valores por teclado y nos muestre posteriormente la suma de los valores ingresados y su promedio.

Diagrama de flujo:



En este problema, a semejanza de los anteriores, llevamos un CONTADOR llamado x que nos sirve para contar las vueltas que debe repetir el while. También aparece el concepto de ACUMULADOR (un acumulador es un tipo

especial de variable que se incrementa o decrementa con valores variables durante la ejecución del programa)

Hemos dado el nombre de suma a nuestro acumulador. Cada ciclo que se repita la estructura repetitiva, la variable suma se incrementa con el contenido ingresado en la variable valor.

La prueba del diagrama se realiza dándole valores a las variables:

valor	suma	x	promedio
	0	1	
(Antes de entrar a la estructura repetitiva estos son los valores) .			
5	5	1	
16	21	2	
7	28	3	
10	38	4	
2	40	5	
20	60	6	
5	65	7	
5	70	8	
10	80	9	
2	82	10	
8	90	11	

9

Este es un seguimiento del diagrama planteado. Los números que toma la variable valor dependerá de qué cifras cargue el operador durante la ejecución del programa. El promedio se calcula al salir de la estructura repetitiva (es decir primero sumamos los 10 valores ingresados y luego los dividimos por 10)

RECORDEMOS APLICACIÓN Y USO DE CICLO WHILE

I. Actividad:

- I.1 Ejecuta los siguientes programas, escribir un programa que solicite ingresar 10 notas de alumnos y nos informe cuántos tienen notas mayores o iguales a 7 y cuántos menores**

LO QUE SABES DEL CICLO DO WHILE



La instrucción do-while permite repetir una instrucción o una instrucción compuesta hasta que una expresión especificada sea false.

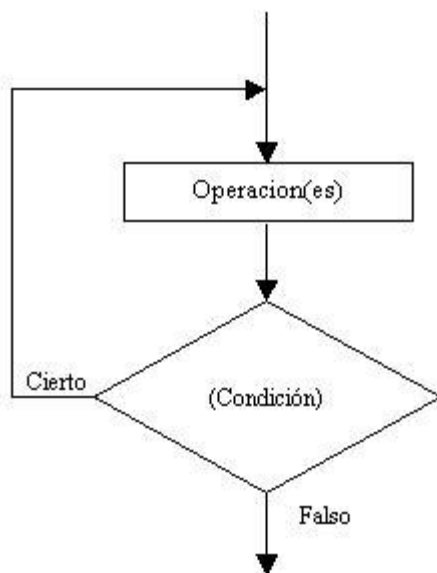
PARA EMPEZAR EL CICLO DO-WHILE

Ciclo DO-WHILE

La estructura do while es otra estructura repetitiva, la cual ejecuta al menos una vez su bloque repetitivo, a diferencia del while o del for que podían no ejecutar el bloque.

Esta estructura repetitiva se utiliza cuando conocemos de antemano que por lo menos una vez se ejecutará el bloque repetitivo. La condición de la estructura está abajo del bloque a repetir, a diferencia del while o del for que está en la parte superior.

Representación gráfica:

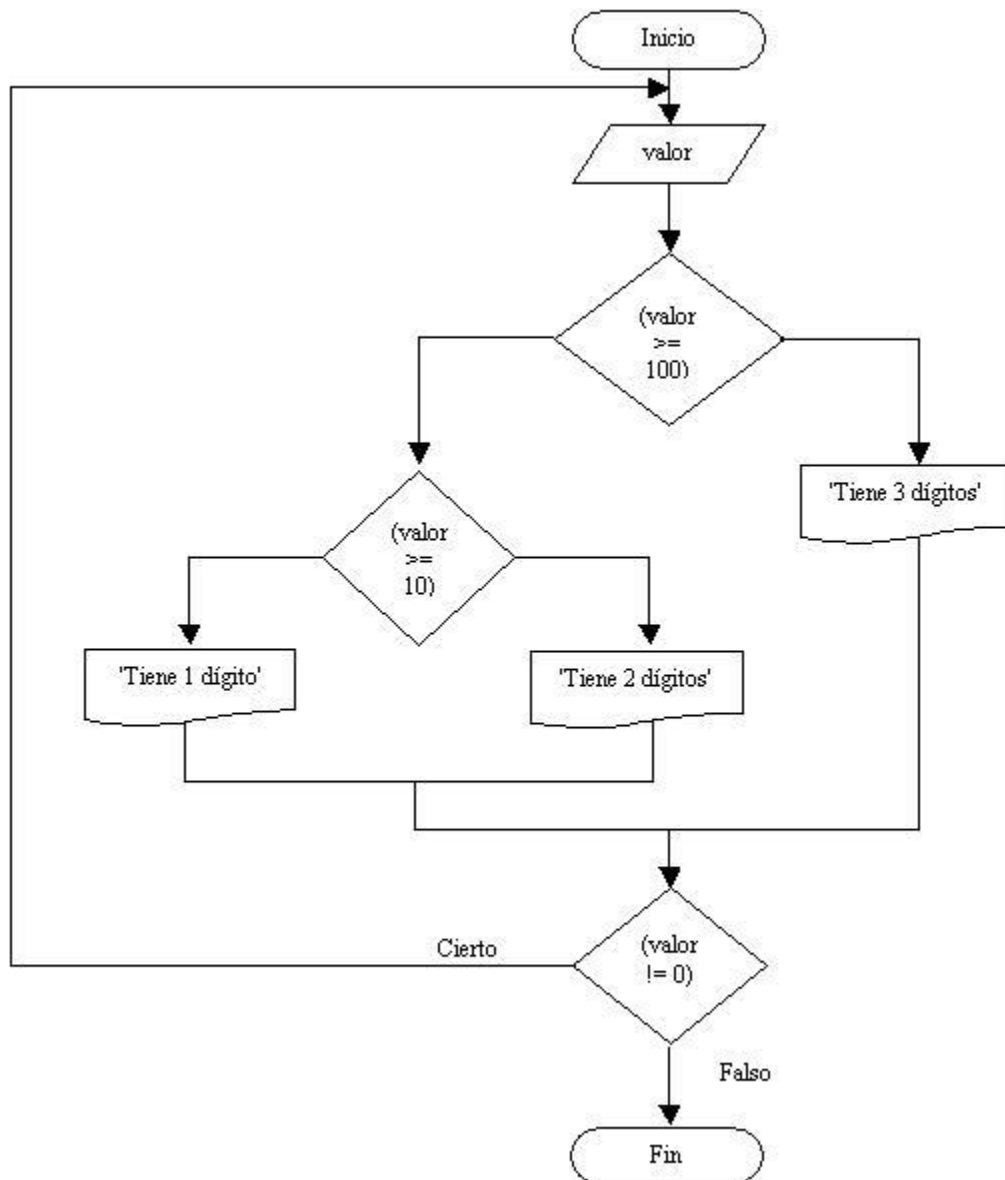


El bloque de operaciones se repite MIENTRAS que la condición sea Verdadera. Si la condición retorna Falso el ciclo se detiene. En C#, todos los ciclos repiten por verdadero y cortan por falso. Es importante analizar y ver que las operaciones se ejecutan como mínimo una vez.

Problema 1:

Escribir un programa que solicite la carga de un número entre 0 y 999, y nos muestre un mensaje de cuántos dígitos tiene el mismo. Finalizar el programa cuando se cargue el valor 0.

Diagrama de flujo:



No hay que confundir los rombos de las estructuras condicionales con los de las estructuras repetitivas `do while`. En este problema por lo menos se carga un valor. Si se carga un valor mayor o igual a 100 se trata de un número de tres cifras, si es mayor o igual a 10 se trata de un

valor de dos dígitos, en caso contrario se trata de un valor de un dígito. Este bloque se repite hasta que se ingresa en la variable valor el número 0 con lo que la condición de la estructura do while retorna falso y sale del bloque repetitivo finalizando el programa.

Programa:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace EstructuraRepetitivaDoWhile1
{
    class Program
    {
        static void Main(string[] args)
        {
            int valor;
            string linea;
            do {
                Console.Write("Ingrese un valor
entre 0 y 999 (0 finaliza):");
                linea = Console.ReadLine();
                valor=int.Parse(linea);
                if (valor>=100)
                {
                    Console.WriteLine("Tiene 3
dígitos.");
                }
                else
                {
                    if (valor>=10)
                    {
                        Console.WriteLine("Tiene 2
dígitos.");
                    }
                    else
                }
            }
        }
    }
}
```



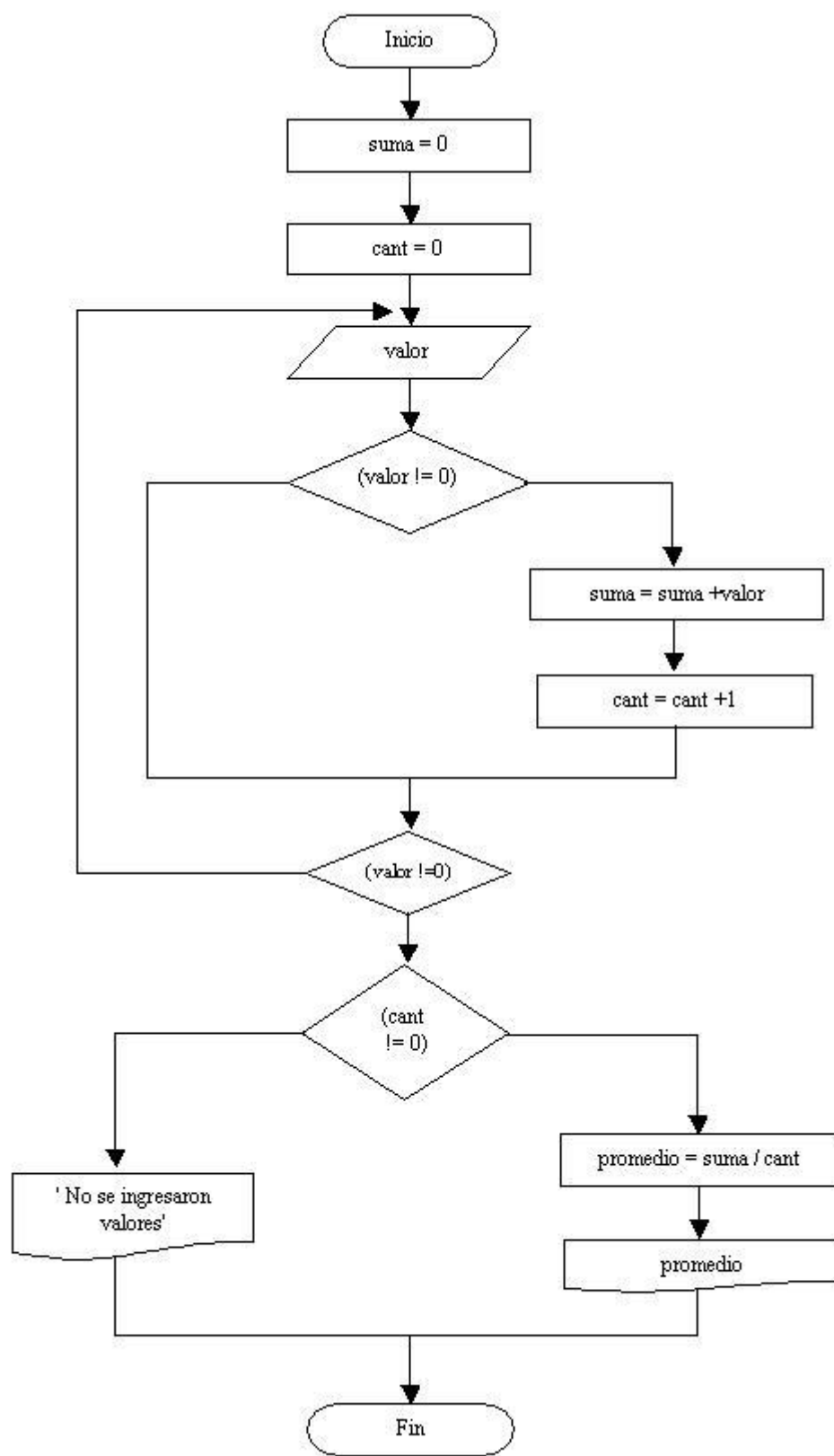
```
        {  
            Console.WriteLine("Tiene 1  
dígito.");  
        }  
    }  
} while (valor!=0);  
}  
}
```

Problema 2:

Escribir un programa que solicite la carga de números por teclado, obtener su promedio. Finalizar la carga de valores cuando se cargue el valor 0.

Cuando la finalización depende de algún valor ingresado por el operador conviene el empleo de la estructura do while, por lo menos se cargará un valor (en el caso más extremo se carga 0, que indica la finalización de la carga de valores)

Diagrama de flujo:



Es importante analizar este diagrama de flujo. Definimos un contador cant que cuenta la cantidad de valores ingresados por el operador (no lo incrementa si ingresamos 0) El valor 0 no es parte de la serie de valores que se deben sumar. Definimos el acumulador suma que almacena todos los valores ingresados por teclado.

La estructura repetitiva do while se repite hasta que ingresamos el valor 0. Con dicho valor la condición del ciclo retorna falso y continúa con el flujo del diagrama. Disponemos por último una estructura condicional para el caso que el operador cargue únicamente un 0 y por lo tanto no podemos calcular el promedio ya que no existe la división por 0. En caso que el contador cant tenga un valor distinto a 0 el promedio se obtiene dividiendo el acumulador suma por el contador cant que tiene la cantidad de valores ingresados antes de introducir el 0.

Programa:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace EstructuraRepetitivaDoWhile2
{
    class Program
    {
        static void Main(string[] args)
        {
            int suma, cant, valor, promedio;
            string linea;
            suma=0;
            cant=0;
            do {
                Console.Write("Ingrese un valor (0
para finalizar):");
                linea = Console.ReadLine();
                valor=int.Parse(linea);
                if (valor!=0) {
                    suma=suma+valor;
```

```
        cant++;  
    }  
} while (valor!=0);  
if (cant!=0) {  
    promedio=suma/cant;  
    Console.Write("El promedio de los  
valores ingresados es:");  
    Console.Write(promedio);  
} else {  
    Console.Write("No se ingresaron  
valores.");  
}  
Console.ReadLine();  
}  
}
```

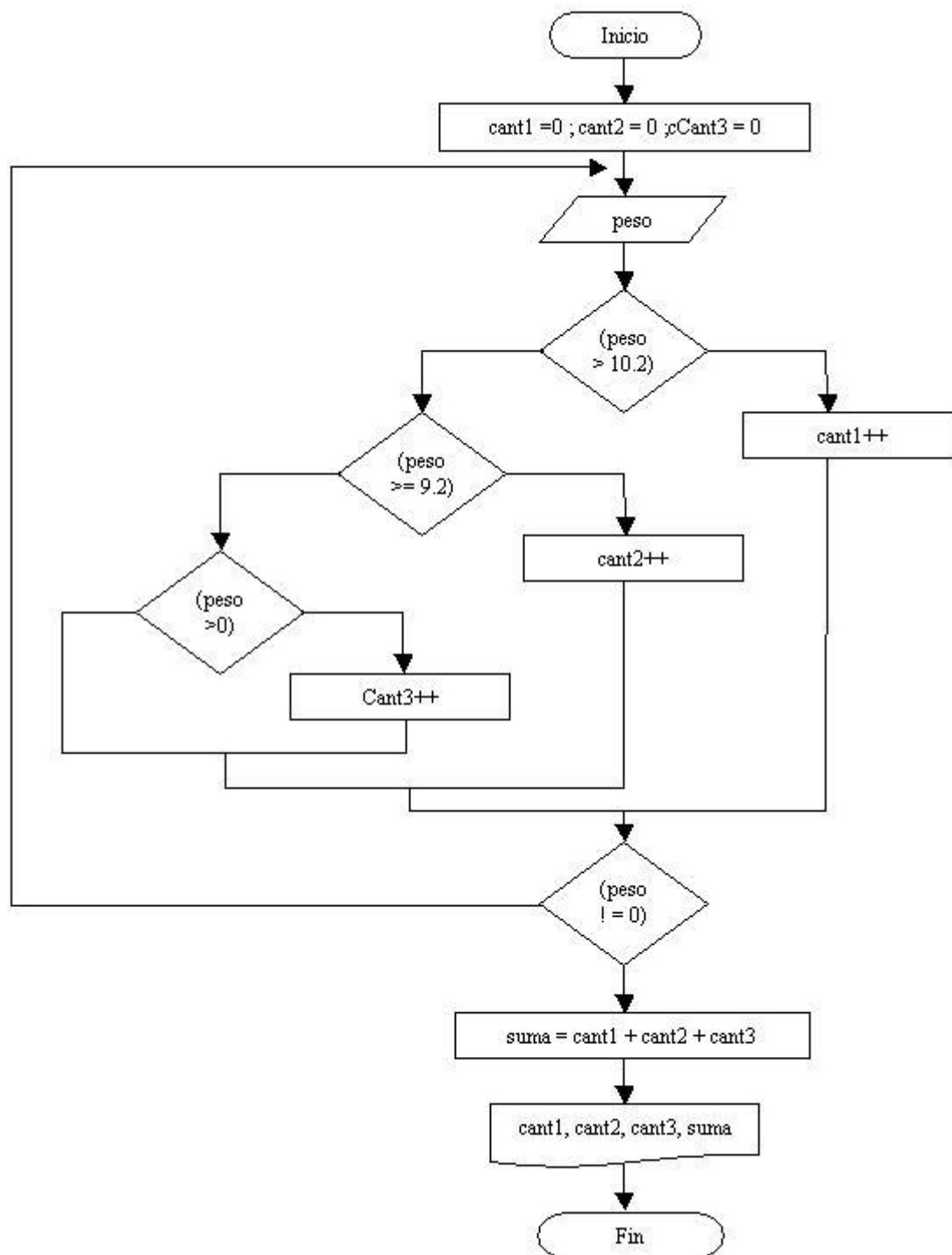
El contador **cant** DEBE inicializarse antes del ciclo, lo mismo que el acumulador **suma**. El promedio se calcula siempre y cuando el contador **cant** sea distinto a 0.

Problema 3:

Realizar un programa que permita ingresar el peso (en kilogramos) de piezas. El proceso termina cuando ingresamos el valor 0. Se debe informar:

- Cuántas piezas tienen un peso entre 9.8 Kg. y 10.2 Kg., cuántas con más de 10.2 Kg. y cuántas con menos de 9.8 Kg.?
- La cantidad total de piezas procesadas.

Diagrama de flujo:



Los tres contadores cant1, cant2, y cant3 se inicializan en 0 antes de entrar a la estructura repetitiva.

A la variable suma no se la inicializa en 0 porque no es un acumulador, sino que guarda la suma del contenido de las variables cant1, cant2 y cant3.

La estructura se repite hasta que se ingresa el valor 0 en la variable peso. Este valor no se lo considera un peso menor a 9.8 Kg., sino que indica que ha finalizado la carga de valores por teclado.

Programa:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace EstructuraRepetitivaDoWhile3
{
    class Program
    {
        static void Main(string[] args)
        {
            int cant1, cant2, cant3, suma;
            float peso;
            string linea;
            cant1=0;
            cant2=0;
            cant3=0;
            do {
                Console.Write("Ingrese el peso de
la pieza (0 pera finalizar):");
                linea = Console.ReadLine();
                peso=float.Parse(linea);
                if (peso>10.2)
                {
                    cant1++;
                }
                else
                {
                    if (peso>=9.8)
                    {
                        cant2++;
                    }
                }
            } while (peso != 0);
        }
    }
}
```

```
        }
        else
        {
            if (peso>0)
            {
                cant3++;
            }
        }
    }
    } while(peso!=0);
    suma=cant1+cant2+cant3;
    Console.Write("Piezas aptas:");
    Console.WriteLine(cant2);
    Console.Write("Piezas con un peso
superior a 10.2:");
    Console.WriteLine(cant1);
    Console.Write("Piezas con un peso
inferior a 9.8:");
    Console.WriteLine(cant3);
    Console.Write("Cantidad de piezas
procesadas:");
    Console.WriteLine(suma);
    Console.ReadLine();
    }
}
}
```

RECORDEMOS USO Y APLICACIÓN DE DO-WHILE

Actividad:

Confecciona el siguiente programa que acumule (sume) valores ingresados por teclado hasta ingresar el 9999 (no sumar dicho valor, indica que ha finalizado la carga). Imprimir el valor acumulado e informar si dicho valor es cero, mayor a cero o menor a cero.

MANOS A LA OBRA CON SEGUNDO PRODUCTO TERCER PARCIAL

Nombre de la práctica: Carrusel de programas

Nomenclatura: MI-SII Practica/ Programas matemáticos

Temas: Ciclo WHILE y DO WHILE

Duración: 2 Sesiones

Objetivos: *Emplear los ciclos WHILE y DO WHILE para la confección de programas que ayuden a resolver problemas de la vida cotidiana.*

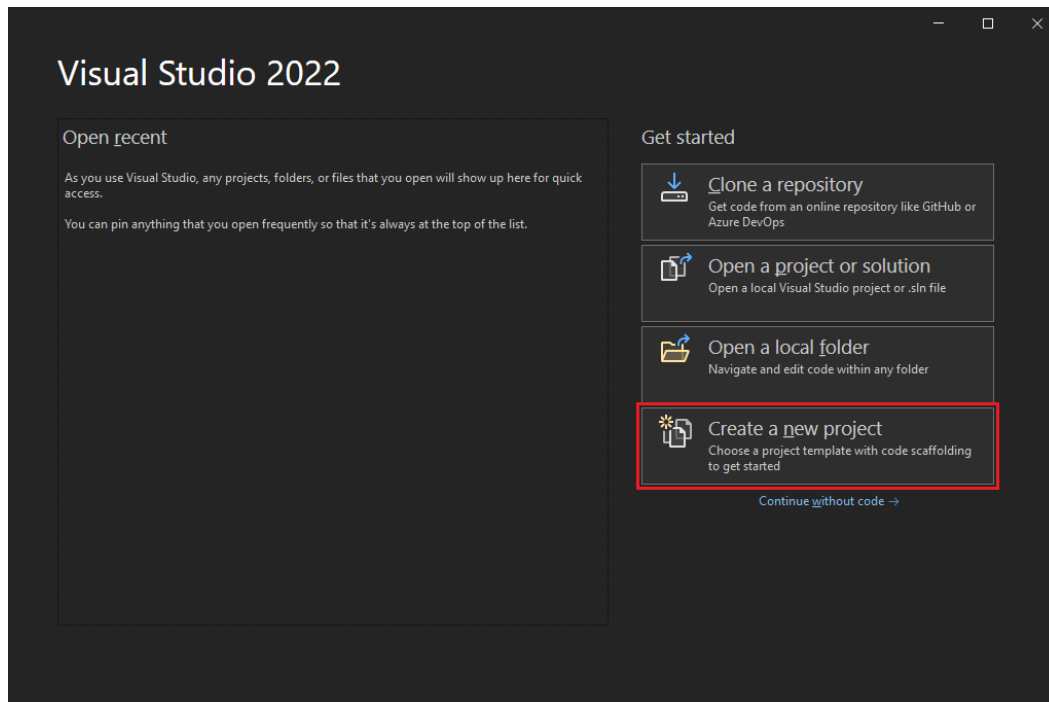
Materiales y equipo: Equipo de cómputo con software instalado Visual Studio o en su caso aplicación de Play Store

Procedimiento:

Crear un proyecto

1. Para empezar, cree un proyecto de aplicación de C#. En el tipo de proyecto se incluyen todos los archivos de plantilla que necesita.

Abra Visual Studio y elija **Crear un nuevo proyecto** en la ventana Inicio.



2. En la ventana **Crear un nuevo proyecto**, seleccione **Todos los lenguajes** y, a continuación, elija **C#** en la lista desplegable. Seleccione **Windows** en la

lista **Todas las plataformas** y **Consola** en la lista **Todos los tipos de proyecto**.

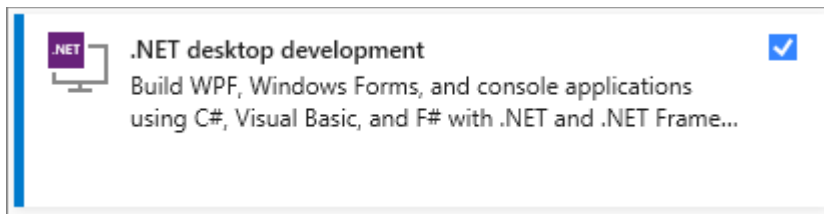
Después de aplicar los filtros de lenguaje, plataforma y tipo de proyecto, elija la plantilla **Aplicación de consola** y, luego, seleccione **Siguiente**.

Nota

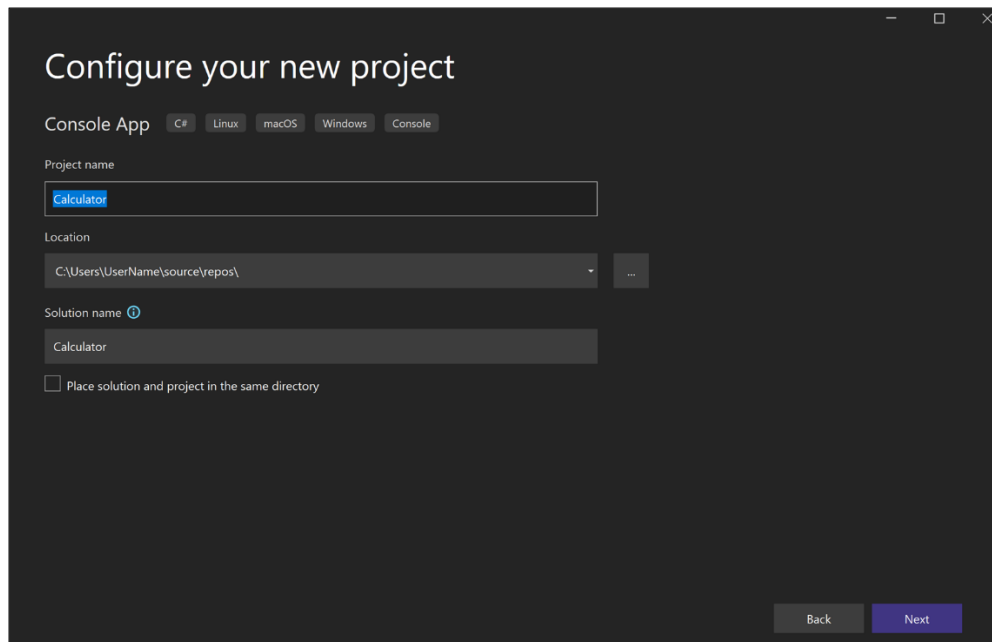
Si no ve la plantilla **Aplicación de consola**, seleccione **Instalar más herramientas y características**.

Not finding what you're looking for?
[Install more tools and features](#)

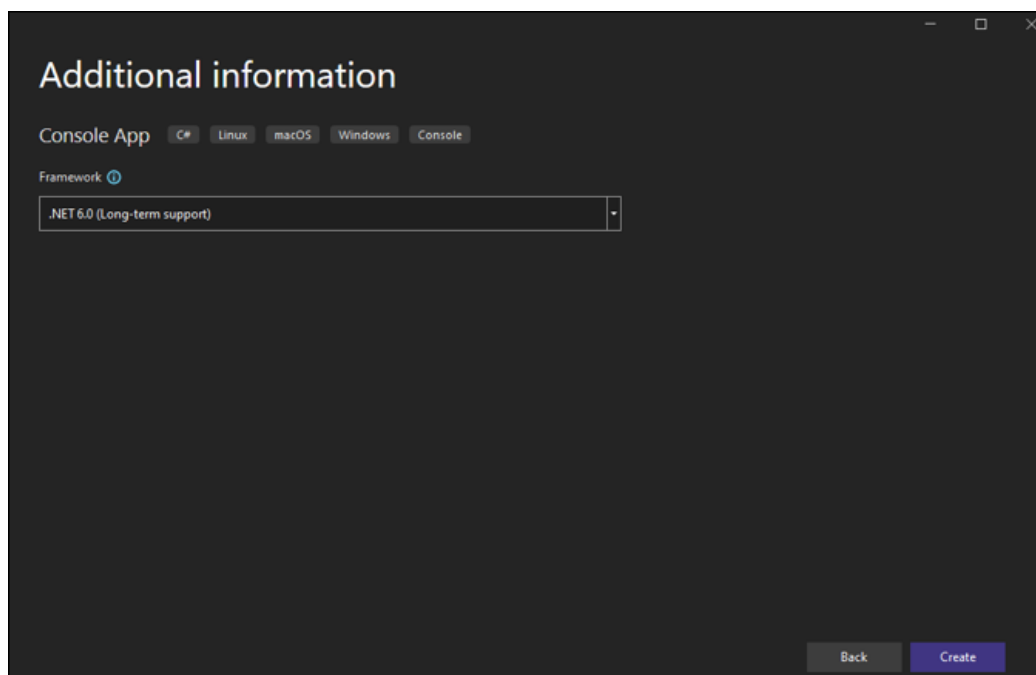
En el Instalador de Visual Studio, elija la carga de trabajo de **desarrollo de escritorio de .NET** y, a continuación, seleccione **Modificar**.



3. En la ventana **Configurar el nuevo proyecto**, escriba CALCULATOR en el cuadro **Nombre del proyecto** y, a continuación, seleccione **Siguiente**.



4. En la ventana **Información adicional**, **.NET 6.0** ya debe estar seleccionado para la plataforma de destino. Seleccione **Crear**.



Visual Studio abre el nuevo proyecto, que incluye código predeterminado de "Hola mundo".

Para verlo en el editor, seleccione el archivo de código Program.cs en la ventana del Explorador de soluciones, que normalmente se encuentra en el lado derecho de Visual Studio.

La instrucción de código única llama al método WriteLine para mostrar la cadena literal "¡Hola mundo!" en la ventana de consola. Si presiona F5, puede ejecutar el programa predeterminado en modo de depuración. Una vez que la aplicación se ejecuta en el depurador, la ventana de la consola permanece abierta. Presione cualquier tecla para cerrar la ventana de consola.

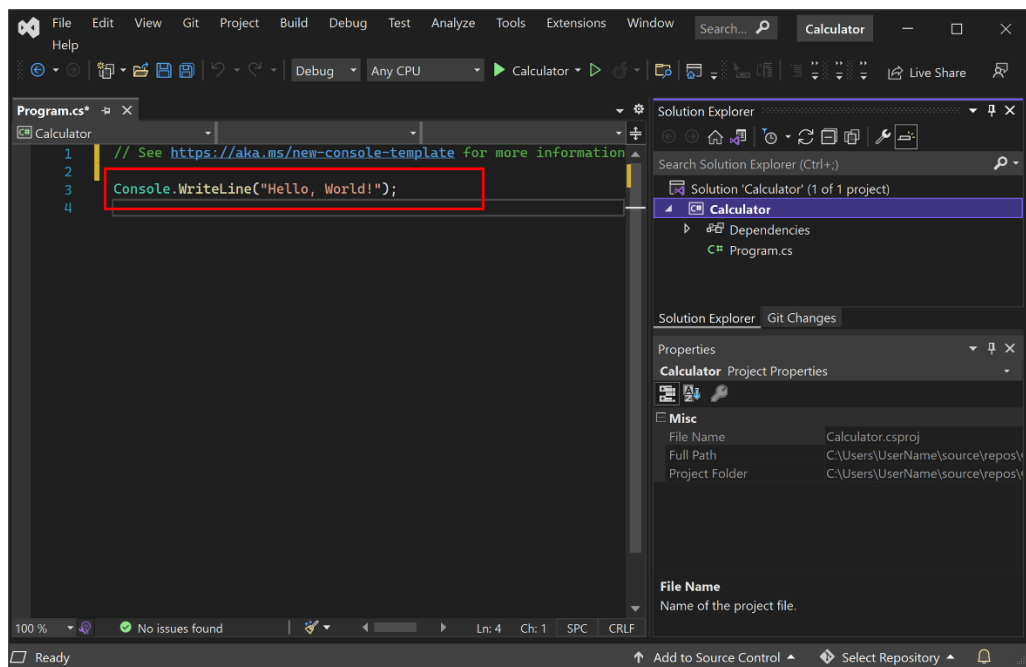
Nota

A partir de .NET 6, los nuevos proyectos que usan la plantilla de consola generan código diferente al de versiones anteriores. Para obtener más información, consulte la página Las nuevas plantillas de C# generan instrucciones de nivel superior.

Creación de la aplicación

Empiece con algunos cálculos básicos de enteros en C#.

5. En el **Explorador de soluciones**, en el panel derecho, seleccione **Program.cs** para mostrar el archivo en el editor de código
6. En el editor de código, reemplace el código predeterminado "Hello World" que dice `Console.WriteLine("Hello World!");`;





7. Ejecuta los siguientes bloques de programas:

- **Escribir un programa que solicite ingresar 10 notas de alumnos y nos informe cuántos tienen notas mayores o iguales a 7 y cuántos menores**
- **Confecciona el siguiente programa que acumule (sume) valores ingresados por teclado hasta ingresar el 9999 (no sumar dicho valor, indica que ha finalizado la carga). Imprimir el valor acumulado e informar si dicho valor es cero, mayor a cero o menor a cero.**

INSTRUMENTO DE EVALUACIÓN

Aspectos a evaluar	Ponderación		
	Porcentaje	Sí	No
Portada con los datos de identificación	2		
Código del programa que solicite ingresar 10 notas de alumnos y nos informe cuántos tienen notas mayores o iguales a 7 y cuántos menores	4		
Resultado del programa que solicite ingresar 10 notas de alumnos y nos informe cuántos tienen notas mayores o iguales a 7 y cuántos menores	4		
Código del programa que solicite ingresar 10 notas de alumnos y nos informe cuántos tienen notas mayores o iguales a 7 y cuántos menores	4		
Resultado del programa que solicite la carga de 10 números e imprima la suma de los últimos 5 valores ingresados	4		
Código del programa que acumule (sume) valores ingresados por teclado hasta ingresar el 9999 (no sumar dicho valor, indica que ha finalizado la carga). Imprimir el valor acumulado e informar si dicho valor es cero, mayor a cero o menor a cero.	4		
Resultado del programa que muestre la tabla de multiplicar del 5 (del 5 al 50)			
Tiene limpieza y formalidad	2		
Se entrega con un folder en condiciones optimas de un trabajo de media superior	2		
Los integrantes trabajan de forma colaborativa	4		
Nombre y firma del docente	Nombre y firma del padre tutor		

LO QUE SABES DE ESTRUCTURA DE CONTROL SWITCH

La estructura condicional switch ... case se utiliza cuando queremos evitarnos las llamadas escaleras de decisiones. La estructura if nos puede proporcionar, únicamente, dos resultados, uno para verdadero y otro para falso. Una estructura switch ... case, por su parte, nos permite elegir entre muchas opciones.

PARA EMPEZAR ANALISIS DE LA ESTRUCTURA SWITCH

Estructura Switch

La estructura condicional switch reemplaza en algunos casos un conjunto de if.

La estructura del switch:

```
switch(variable) {  
    case valor1:  
        Instrucciones  
        break;  
    case valor2:  
        Instrucciones  
        break;  
    case valor3:  
        Instrucciones  
        break;  
    .  
    .  
    .  
    default:  
        Instrucciones  
        break;  
}
```

Luego de la palabra clave switch entre paréntesis indicamos una variable, con una serie de case verificamos si dicha variable almacena un valor igual a [valor1, valor2, valor3 etc.] en el caso de ser igual se ejecutan las instrucciones contenidas en dicho case.

Si todos los case son falsos, luego se ejecutan las instrucciones contenidas después de la palabra default.

PROBLEMA 1:

Ingresar un valor entero entre 1 y 5. Luego mostrar en castellano el valor ingresado. Si se ingresa un valor fuera de dicho rango mostrar un mensaje indicando tal situación

PROGRAMA:

```
using System;  
using System.Collections.Generic;  
using System.Linq;
```

```
using System.Text;

namespace Estructuraswitch1
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.Write("Ingrese un valor entre
1 y 5:");
            int valor =
int.Parse(Console.ReadLine());
            switch (valor)
            {
                case 1: Console.Write("uno");
                    break;
                case 2: Console.Write("dos");
                    break;
                case 3: Console.Write("tres");
                    break;
                case 4: Console.Write("cuatro");
                    break;
                case 5: Console.Write("cinco");
                    break;
                default:
                    Console.Write("Se ingreso
un valor fuera de rango");
                    break;
            }
            Console.ReadKey();
        }
    }
}
```

Es obligatorio que esté entre paréntesis la variable luego de la palabra clave switch. Luego de cada case debemos indicar el valor con el que se comparará la variable (siempre debe ser un valor constante y no podemos disponer una variable luego de la palabra case).

Es necesario la palabra break luego de cada bloque de instrucciones por cada case.

PROBLEMA 2:

Ingresar un número entre uno y cinco en castellano. Luego mostrar en formato numérico. Si se ingresa un valor fuera de dicho rango mostrar un mensaje indicando tal situación

PROGRAMA:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Estructuraswitch2
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.Write("Ingrese un número en
castellano entre uno y cinco:");
            string nro = Console.ReadLine();
            switch (nro)
            {
                case "uno": Console.Write(1);
                    break;
                case "dos": Console.Write(2);
                    break;
                case "tres": Console.Write(3);
                    break;
                case "cuatro": Console.Write(4);
                    break;
                case "cinco": Console.Write(5);
                    break;
                default: Console.Write("Debe
ingresar un valor entre uno y cinco");
                    break;
            }
            Console.ReadKey();
        }
    }
}
```

}

RECORDEMOS USO Y APLICACIÓN DE ESTRUCTURA SWITCH

Actividad

- I. **Confecciona los programas propuestos en la sesión y modificalos según las necesidades de representar información**

MANOS A LA OBRA CON TERCER PRODUCTO TERCER PARCIAL

Nombre de la práctica: Juego de opciones

Nomenclatura: MI-SII Practica/ Juego de opciones

Temas: Ciclo SWITCH

Duración: 3 Sesiones

Objetivos: *Emplear los ciclos SWITCH para la confección de programas que ayuden a resolver problemas de la vida cotidiana.*

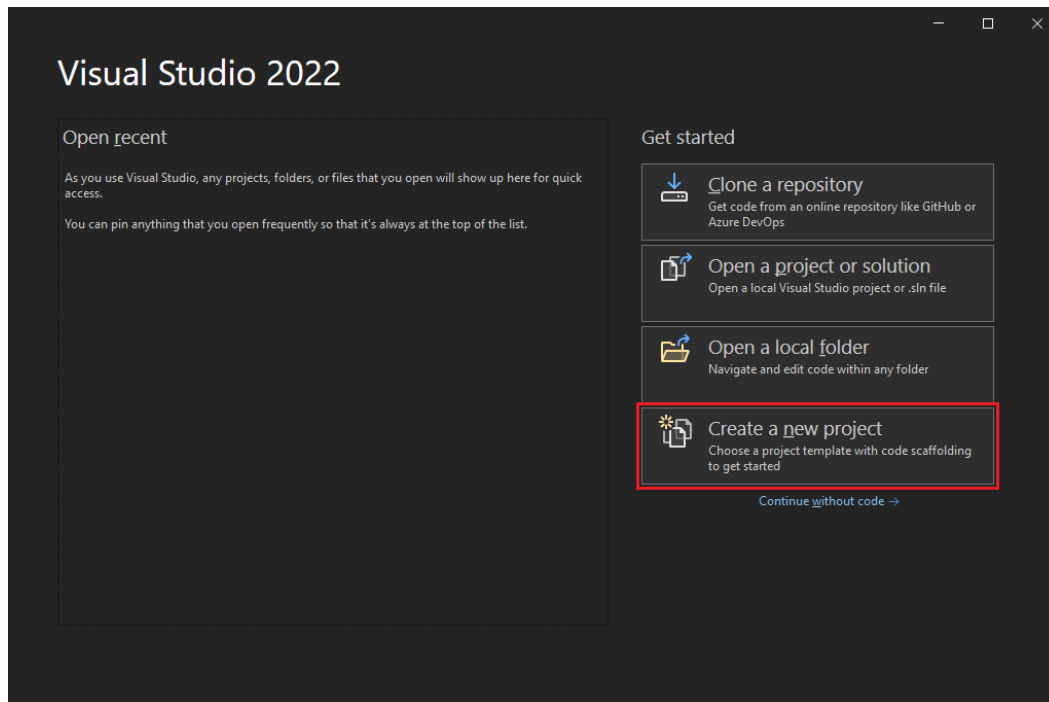
Materiales y equipo: Equipo de cómputo con software instalado Visual Studio o en su caso aplicación de Play Store

Procedimiento:

Crear un proyecto

- I. Para empezar, cree un proyecto de aplicación de C#. En el tipo de proyecto se incluyen todos los archivos de plantilla que necesita.

Abra Visual Studio y elija **Crear un nuevo proyecto** en la ventana Inicio.



2. En la ventana **Crear un nuevo proyecto**, seleccione **Todos los lenguajes** y, a continuación, elija **C#** en la lista desplegable. Seleccione **Windows** en la lista **Todas las plataformas** y **Consola** en la lista **Todos los tipos de proyecto**.

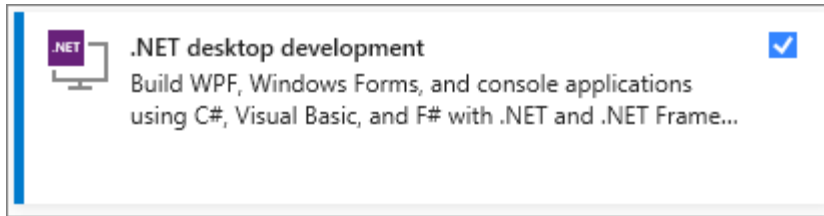
Después de aplicar los filtros de lenguaje, plataforma y tipo de proyecto, elija la plantilla **Aplicación de consola** y, luego, seleccione **Siguiente**.

Nota

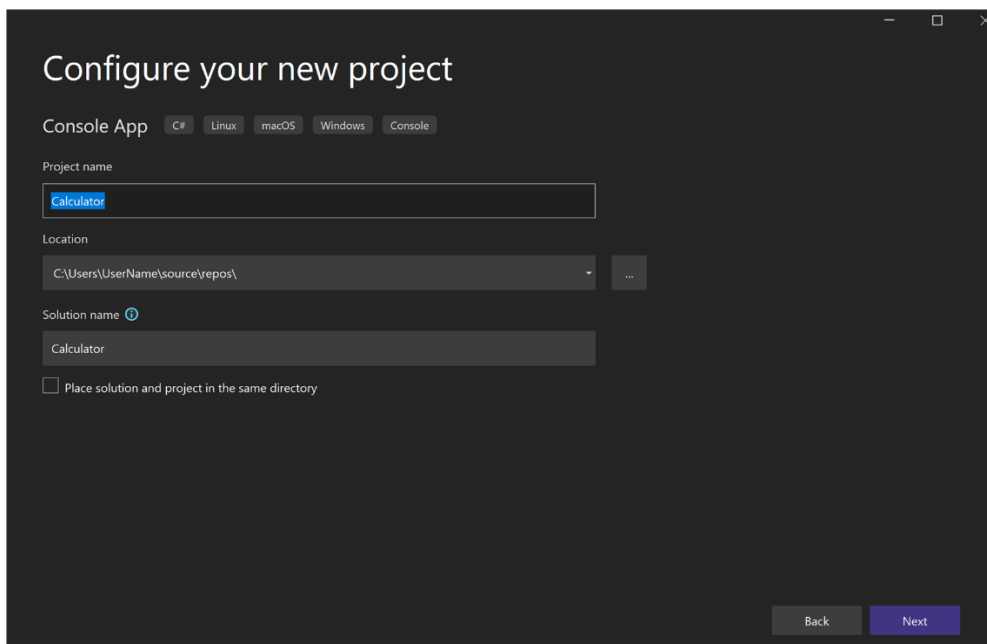
Si no ve la plantilla **Aplicación de consola**, seleccione **Instalar más herramientas y características**.

Not finding what you're looking for?
[Install more tools and features](#)

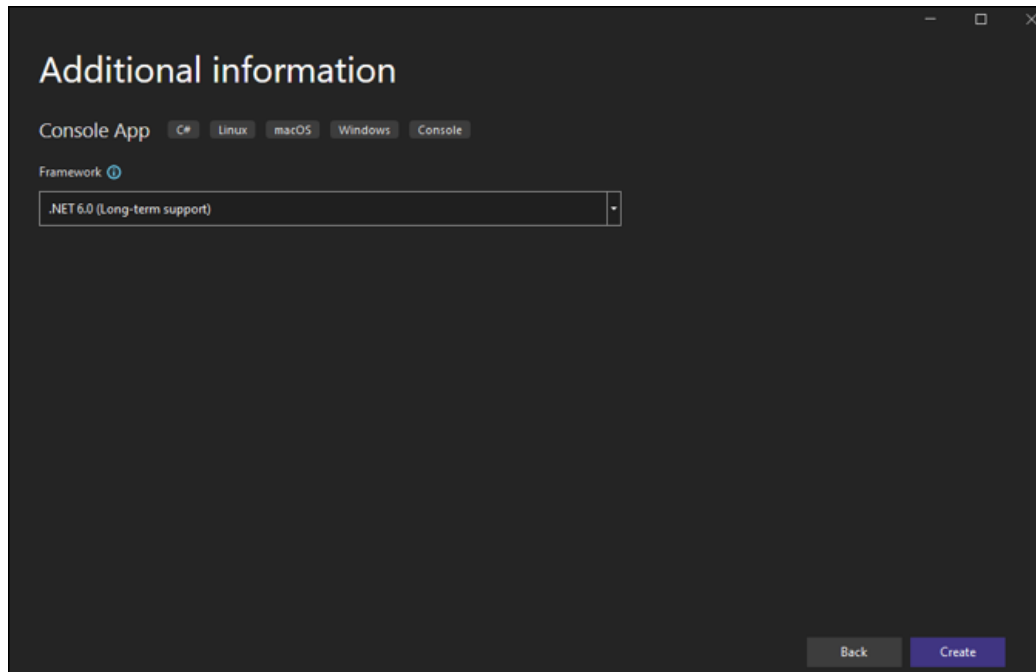
En el Instalador de Visual Studio, elija la carga de trabajo de **desarrollo de escritorio de .NET** y, a continuación, seleccione **Modificar**.



3. En la ventana **Configurar el nuevo proyecto**, escriba CALCULATOR en el cuadro **Nombre del proyecto** y, a continuación, seleccione **Siguiente**.



4. En la ventana **Información adicional**, **.NET 6.0** ya debe estar seleccionado para la plataforma de destino. Seleccione **Crear**.



Visual Studio abre el nuevo proyecto, que incluye código predeterminado de "Hola mundo".

Para verlo en el editor, seleccione el archivo de código Program.cs en la ventana del Explorador de soluciones, que normalmente se encuentra en el lado derecho de Visual Studio.

La instrucción de código única llama al método WriteLine para mostrar la cadena literal "¡Hola mundo!" en la ventana de consola. Si presiona F5, puede ejecutar el programa predeterminado en modo de depuración. Una vez que la aplicación se ejecuta en el depurador, la ventana de la consola permanece abierta. Presione cualquier tecla para cerrar la ventana de consola.

Nota

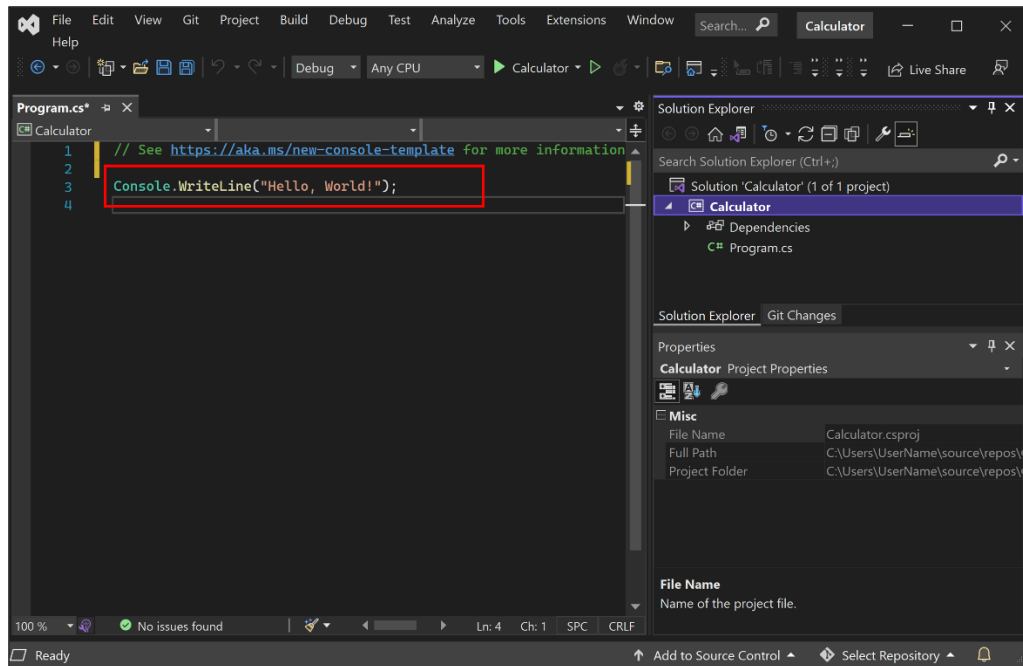
A partir de .NET 6, los nuevos proyectos que usan la plantilla de consola generan código diferente al de versiones anteriores. Para obtener más información, consulte la página Las nuevas plantillas de C# generan instrucciones de nivel superior.

Creación de la aplicación

Empiece con algunos cálculos básicos de enteros en C#.

5. En el **Explorador de soluciones**, en el panel derecho, seleccione **Program.cs** para mostrar el archivo en el editor de código

- En el editor de código, reemplace el código predeterminado "Hello World" que dice `Console.WriteLine("Hello World!");`;



- Ejecuta los siguientes bloques de programas:

- **Confecciona los programas vistos en la sesión**

INSTRUMENTO DE EVALUACIÓN

Aspectos a evaluar	Ponderación		
	Porcentaje	Sí	No
Portada con los datos de identificación	2		
Código del programa que Ingresar un valor entero entre 1 y 5. Luego mostrar en castellano el valor ingresado. Si se ingresa un valor fuera de dicho rango mostrar un mensaje indicando tal situación	2		
Resultado del programa que Ingresar un valor entero entre 1 y 5. Luego mostrar en castellano el valor ingresado. Si se ingresa un valor fuera de dicho rango mostrar un mensaje indicando tal situación	2		
Código del programa que solicite ingresar 10 notas de alumnos y nos informe cuántos tienen notas mayores o iguales a 7 y cuántos menores	2		
Resultado del programa que solicite Ingresar un número entre uno y cinco en castellano. Luego mostrar en formato numérico. Si se ingresa un valor fuera de dicho rango mostrar un mensaje indicando tal situación	2		
Código del programa que solicite Ingresar un número entre uno y cinco en castellano. Luego mostrar en formato numérico. Si se ingresa un valor fuera de dicho rango mostrar un mensaje indicando tal situación	2		
Tiene limpieza y formalidad	2		
Se entrega con un folder en condiciones optimas de un trabajo de media superior	2		
Los integrantes trabajan de forma colaborativa	4		
Nombre y firma del docente	Nombre y firma del padre tutor		



GLOSARIO

RECURSOS DE APOYO