Desarrollo de programas



## Desarrollo de programas

Dos ideas previas sencillas:

- El objetivo de un programa es resolver un problema.
- El ordenador *no piensa*, solo ejecuta un programa dado.

Un programa tiene que estar bien escrito de acuerdo con la sintaxis del lenguaje, para que pueda ser ejecutado.

Pero además debe ser correcto

¿Qué significa que un programa sea correcto?

# Programas correctos (I)

Un programa es correcto si hace exactamente la tarea para la que ha sido diseñado.

Formalmente: es correcto si se comporta exactamente según una especificación dada. La especificación define o describe lo que debe hacer hacer el programa.

A su vez la especificación puede ser:

- Formal (utilizando lenguajes formales de especificación, que utilizan la lógica matemática). Por ejemplo, con el modelo de precondición y postcondición y la lógica de Hoare.
- Menos formal, utilizando lenguaje natural.

En cualquier caso la especificación debe ser absolutamente precisa a la hora de describir qué debe hacer el programa.



# Programas correctos (II)

En particular, un programa correcto:

- Debe funcionar para todos los casos previstos, dando el resultado esperado según la especificación
- Debe estar completamente libre de errores.
- Debe terminar adecuadamente

Además, la buena metodología de programación busca que los programas:

- Estén bien estructurados (bloques de código bien organizados)
- Sean eficientes en tiempo de ejecución y en consumo de memoria (en general, en consumo de recursos).
- Estén escritos de forma clara y bien documentados. Con ello se consigue que sean más fáciles de entender, reutilizar y mantener.

# Desarrollo de programas. Metodología.

1. Análisis y Especificación del problema (¿QUÉ?):
   - extraer información relevante, eliminar ambigüedades del planteamiento
   - identificar los datos de entrada o input y los datos de salida o output y qué hay que hacer
2. Diseño de un algoritmo (¿CÓMO?): precisar los pasos para obtener la solución requerida (output) a partir de la entrada (input).
   - Partir de planteamiento general prescindiendo de detalles (dejar pendientes subproblemas más pequeños). Después abordar estos subproblemas con la misma técnica $\rightsquigarrow$ Diseño descendente o divide y vencerás o aproximación por refinamientos sucesivos.
3. Implementar el algoritmo en un lenguaje concreto (como C#, en nuestro caso). Compilarlo, corregir posibles errores de sintáxis, ...
4. Probar y depurar (test): comprobar el funcionamiento del programa con una batería de ejemplos *intentando* cubrir toda la casuística posible.
   - Otra alternativa verificación formal de la corrección del algoritmo $\rightsquigarrow$ demostración formal (matemática) utilizando la lógica de Hoare, verificadores (semi)-automáticos de programas...



# Desarrollo de programas (II)

Es habitual completar las 4 fases anteriores e iterar, es decir, volver a la fase 1, repasar la especificación (a veces el diseño del algoritmo o la propia implementación requieren modificar la especificación), adaptar la implementación, etc....
En en el *ciclo de vida de un programa*, puede incluirse la fase 5:

- Mantenimiento: modificaciones y actualizaciones de programa para satisfacer nuevos requisitos o aumentar prestaciones (o corregir errores no detectados).

# Ejemplo (I)

Problema:

*averiguar el más pequeño entre dos números dados en cualquier orden*

Primer paso, **Análisis y especificación**:

- Información irrelevante?: "dados en cualquier orden"
- Ambigüedad?: ¿cuál es el más pequeño entre 6 y 6?...
  - suponemos que el usuario (cliente) desea 6 como respuesta (o se lo preguntamos para aclararlo)
- Imprecisión?: los números dados, ¿son naturales, enteros, reales, complejos?...
  - supondremos que son enteros (o preguntamos al cliente)



# Ejemplo (II)

**Especificación** (distintas formas de hacerla):

- en lenguaje natural (español), pero precisa:

  *determinar el mínimo entre dos enteros dados*

- más formal, apoyada en lenguaje lógico/matemático:

  dados $x, y \in \mathbb{Z}$ determinar $z = min(x, y)$ siendo

  $$min(x,y) = \begin{cases} x & \text{si } x \leq y \\ y & \text{en otro caso} \end{cases}$$

- completamente formal, p.e., en estilo funcional:
  se nos pide una función $f : (\mathbb{Z}, \mathbb{Z}) \to \mathbb{Z}$ que verifique:

  $$f(x,y) = \begin{cases} x & \text{si } x \leq y \\ y & \text{en otro caso} \end{cases}$$

Identificamos entrada y salida (input/output):

- input: dos números enteros, $x, y$
- output: el más pequeño de ellos, mínimo entre $x$ y $y$.

# Ejemplo (III)

Segundo paso, diseño del algoritmo: secuencia de acciones a realizar.

- Aproximación I:

  *solicitar de teclado los números de entrada $x$ e $y$*
  *calcular en $z$ el mínimo de $x$ e $y$*
  *escribir $z$ en pantalla*

- Aproximación II (refinamiento de I):

  *solicitar de teclado los números de entrada $x$ e $y$*
  *si $x \leq y$ hacer $z = x$*
  *en caso contrario hacer $z = y$*
  *escribir $z$ en pantalla*



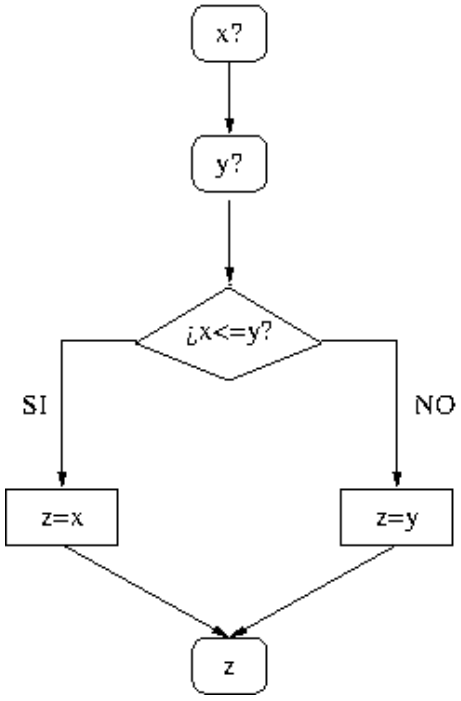
# Ejemplo (III)

- Aproximación III (refinamiento de II):

  *escribir 'x? ' en pantalla*
  *leer el valor de $x$ de teclado*
  *escribir 'y? ' en pantalla*
  *leer el valor de $y$ de teclado*
  *si $x \leq y$ hacer $z = x$*
  *en caso contrario hacer $z = y$*
  *escribir 'z =' en pantalla*
  *escribir el valor de $z$ en pantalla*

En este nivel de refinamiento se ha detallado suficientemente el algoritmo, si no, se continuaría.

Diagramas de flujo para algoritmos: otra forma de presentar los algoritmos (no la utilizaremos habitualmente):






# Ejemplo (IV)

Ahora habría que hacer la implementación en un lenguaje concreto. Por ejemplo, en Python se podría hacer como (ahora no nos importan los detalles del lenguaje):

```
print "Primer entero: ",
x = int(input())
print "Segundo entero: ",
y = int(input())
if x <= y:
    z = x
else:
    z = y
print "El menor es: ", z
```

# Ejemplo (V)

Quinto paso, **test**.

```
Primer entero:  8
Segundo entero:  -17
El menor es:  -17
```

Esto es simplemente un test, no un prueba de corrección.

**Verificación formal**: precondición $\xrightarrow{programa}$ postcondición

$\{P_0 \equiv x, y, z \in \mathbb{Z}\}$

```
if (x<=y) then z:=x
else z:=y
```

$\{P_1 \equiv z = min(x, y)\}$

La **lógica de Hoare** define el comportamiento de cada instrucción.



# Ejemplo (VI)

**Mantenimiento**. El usuario pide solicita cambios:

- Cambio en la especificación: los números en vez de enteros que sean reales (ampliación de la funcionalidad).
- Cambios de petición y presentación de datos: que pida los dos números a la vez, que escriba en salida también los números de entrada.

En general, en grandes programas a veces hay pequeños (o grandes) errores. Una parte del mantenimiento también consiste en corregir estos errores.

Esta secuencia es lo que habitualmente se denomina *ciclo de vida del Software*.

# Otro ejemplo

Problema:

*calcular la suma de los n primeros naturales*

- **Análisis y especificación:**

  dado $n \in \mathbb{N}$ evaluar $1 + 2 + \ldots + n$ (en términos matemáticos $\Sigma_{i=1}^{n} i$)

  input: $n \in \mathbb{N}$; output: $1 + 2 + \ldots + n$

- **Algoritmo:** (sin aplicar la fórmula conocida)
  Aproximación I:

  *solicitar de teclado el valor n*
  *calcular* $Suma = 1 + 2 + \ldots + n$
  *escribir Suma en pantalla*



# Refinamiento

Refinamiento I (sólo del cálculo en sí):

*inicializar* $Suma = 0$
*repetir desde* $i = 1$ *hasta n*
$Suma = Suma + i$

**Implementación** (ahora en Pascal):

```
{Este programa calcula la suma...}
program sumatorio;
var i,n,suma: integer;
begin
    write('valor de n: '); readln(n);
    suma:=0;              {inicialización del acumulador}
    for i:=1 to n do      {se va incrementando el acumulador}
        suma:=suma+i;     {sumando los valores de i}
    writeln('la suma es: ', suma);
end.
```

Luego depuración, mantenimiento...

# Compiladores



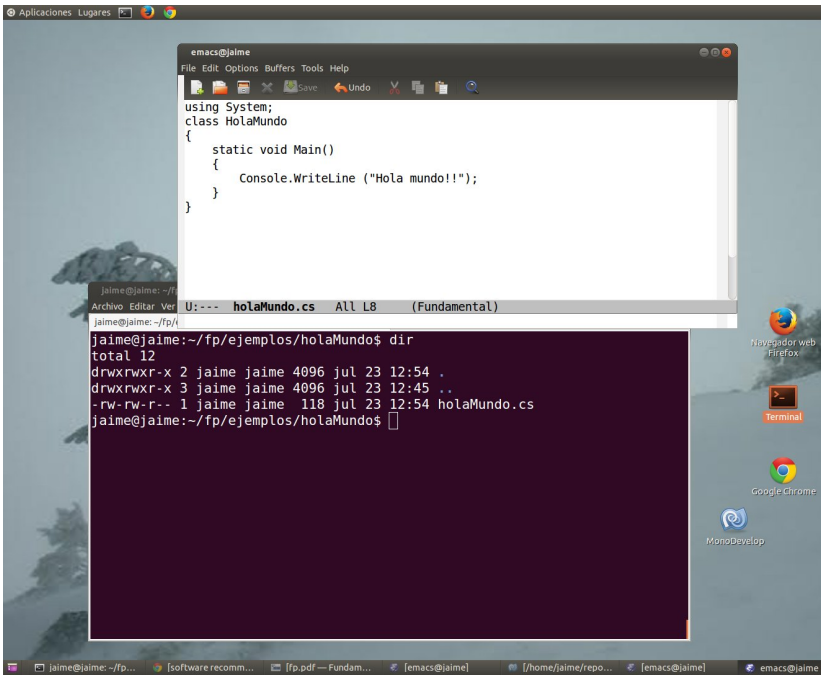
## Nuestro primer programa en C#: "hola mundo!"

```
using System;
class HolaMundo
{
    static void Main()
    {
     Console.WriteLine ("Hola mundo!!");
    }
}
```

No es necesario entender este programa por ahora...

- Lo escribimos en nuestro editor de texto favorito (emacs, gedit, notepad, ...)
- Lo guardamos en un archivo `holaMundo.cs`
- Cómo lo ejecutamos?

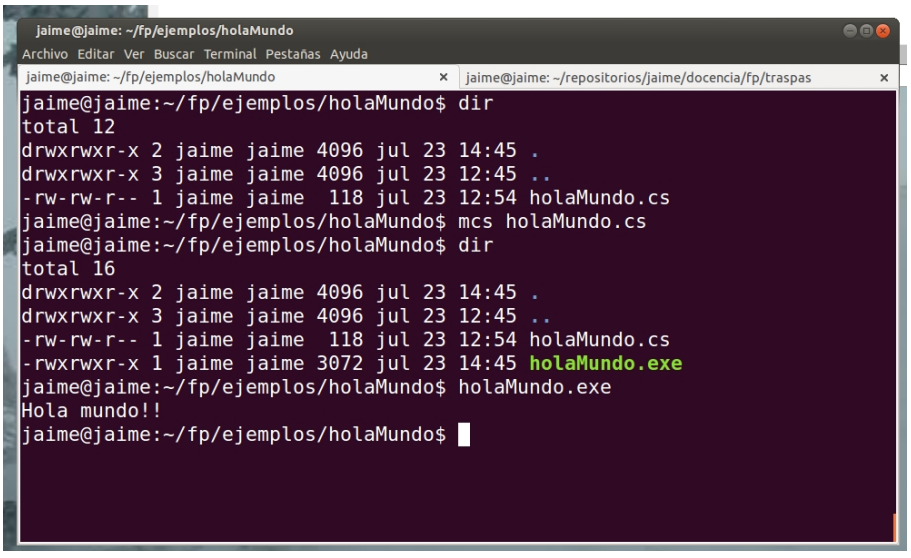
# Compilando desde línea de comandos (I)




La línea de comandos todavía existe! (en Linux, Windows, Mac...)



# Compilando desde línea de comandos (II)




- Compilación desde línea de comandos (desde un terminal):
  `> mcs holaMundo.cs`
- Esto produce un ejecutable `holaMundo.exe`
- Para ejecutar el programa, desde línea de comandos:
  `> holaMundo.exe`
- Y produce el resultado esperado... acabamos de escribir, compilar y ejecutar nuestro primer programa!

# Pero...

- ¿Qué significa exactamente compilar un programa?
- ¿Qué es un compilador?
- ¿Qué significa ejecutar un programa?
- ¿Quién compila?
- ¿Quién ejecuta?



# ¿Qué significa exactamente compilar un programa?

Compilar es traducir: se traduce el código fuente escrito en un lenguaje de programación (como C#) código objeto. Este código objeto:

- puede ser código binario para una máquina real (CPU) directamente ejecutable por la misma (ceros y unos que "entiende" el ordenador, i.e., instrucciones para el microprocesador)
- o puede ser código para una máquina virtual (bytecode), que puede ser fácilmente convertible en código ejecutable de manera eficiente.

Un compilador es un programa que traduce un programa escrito en un lenguaje de programación a otro lenguaje, para poder ejecutarlo en el ordenador.

En concreto, C# se compila/traduce a un lenguaje intermedio para máquina virtual (Common Intermediate Language CIL), utilizado en la plataforma .NET (usamos el compilador **mcs**).

## ¿Qué significa ejecutar un programa? ¿Quién ejecuta?

- Ejecutar un programa es hacerlo funcionar en el ordenador. Para ello, el sistema operativo (linux, windows,...) lo carga en memoria y la CPU efectúa las instrucciones de ese programa.
- El sistema operativo (SO) es el que ejecuta los programas en el ordenador.
- El sistema operativo, a su vez es un programa de base (kernel del SO) que utiliza a su vez un conjunto de programas (más o menos básicos).



## Lenguajes de programación y compiladores

• Las ideas de compilador y lenguaje de programación están muy ligadas, pero no son lo mismo:

- En general puede haber distintos compiladores para un mismo lenguaje de programación. Por ejemplo, para el lenguaje C# hay varios compiladores como *MS VisualStudio* o *monodevelop*.

• Otro tipo de programas muy relacionados con los compiladores son los intérpretes. También hacen una traducción de lenguaje fuente a objeto, pero en este caso el programa se va ejecutando a medida que se hace la traducción, mientras que el compilador hace toda la traducción y genera el código objeto (ejecutable), pero no hace la ejecución misma.

# Entornos de desarrollo

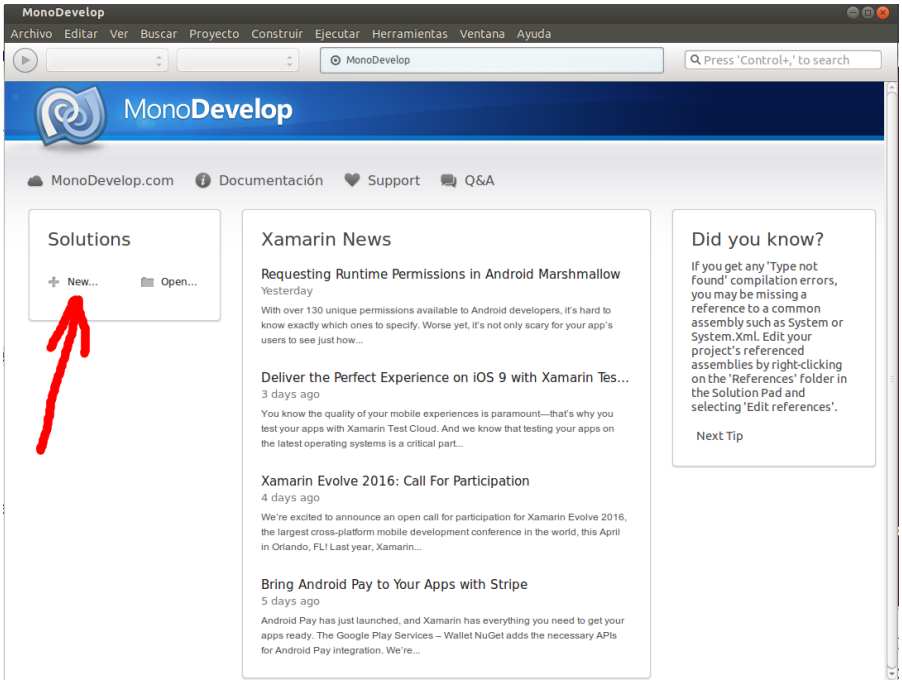
- En la actualidad es muy frecuente que los compiladores se integren en entornos integrados de desarrollo: entorno gráfico para el desarrollo de programas que incluye:
  - Editor de texto:
    - con resaltado de sintaxis, auto-completado inteligente de código, herramientas de construcción automáticas (plantillas de programa, etc).
  - El compilador propiamente dicho. No se lanza desde línea de comandos, sino con botones (a golpe de click de ratón).
  - Otras herramientas adaptadas al lenguaje concreto: enlazador de librerías, navegador de clases, ayuda sensitiva, etc.
  - Depurador de código (debugger). Es una herramienta que permite hacer trazas de ejecución del programa: ejecutar el programa *paso a paso* (seguir el flujo de ejecución) viendo en el contenido de las variables, la pila de ejecución, etc.

Es un gran aliado para el programador para detectar y corregir errores.



# El entorno MonoDevelop para C#

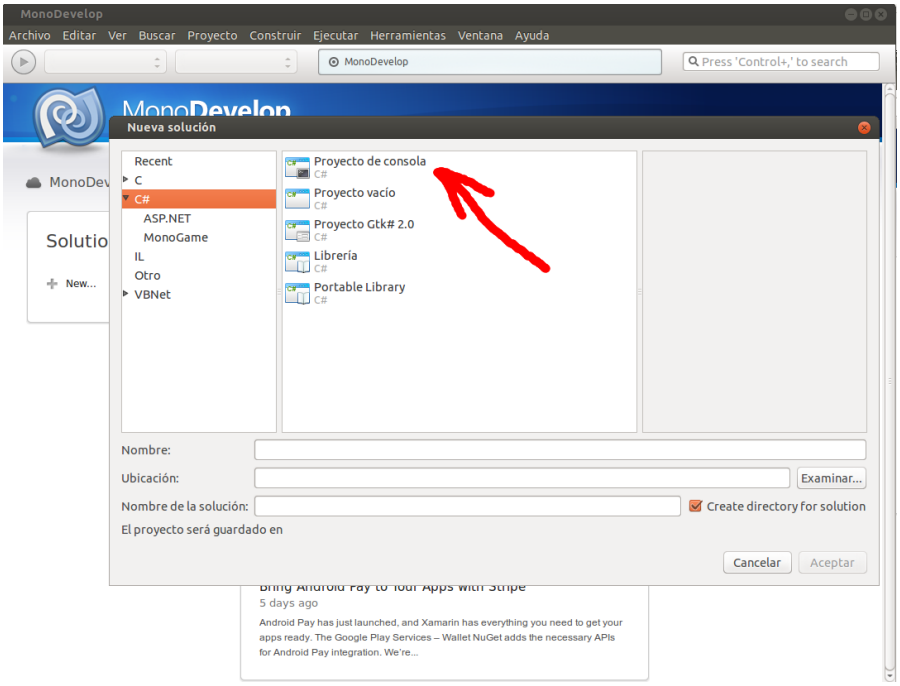
Ventana de inicio




Generamos una nueva solución

# El entorno MonoDevelop para C#

Nuestra solución, en este caso, será un proyecto de consola

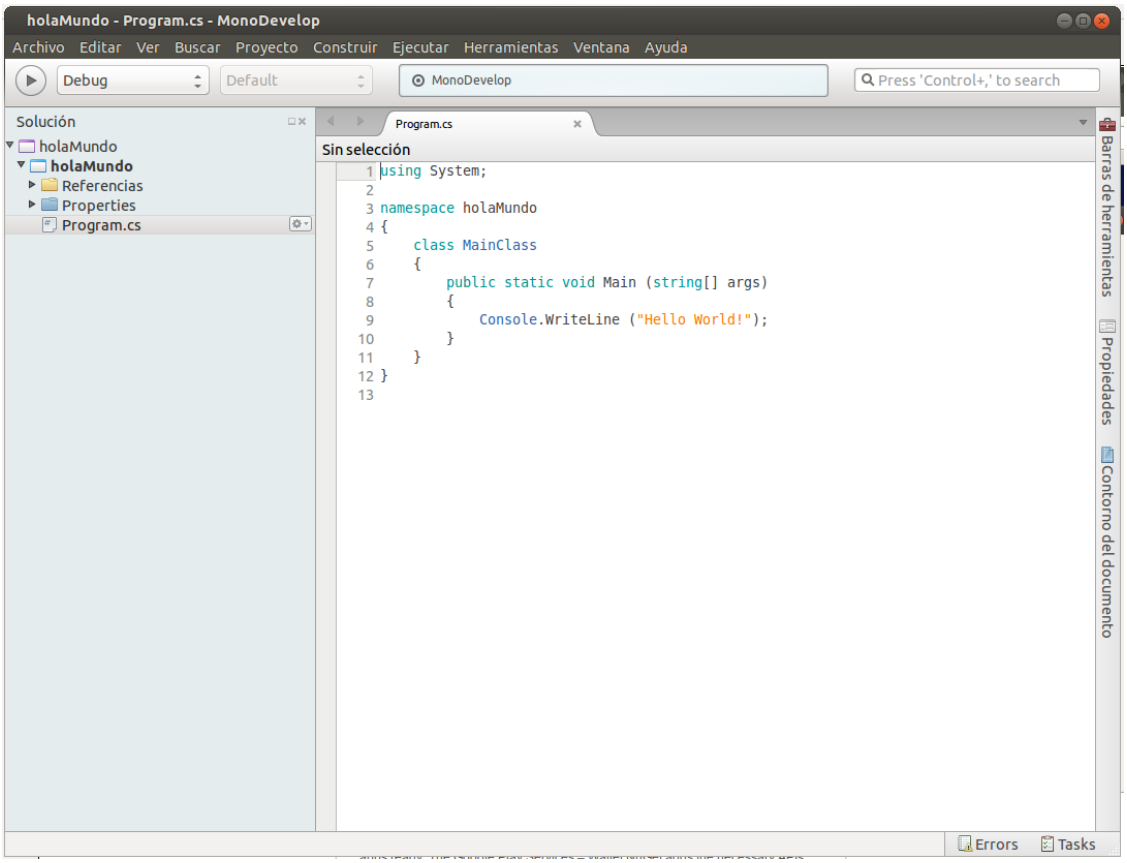



Tenemos que dar nombre al proyecto, a la solución y una ubicación en disco duro.



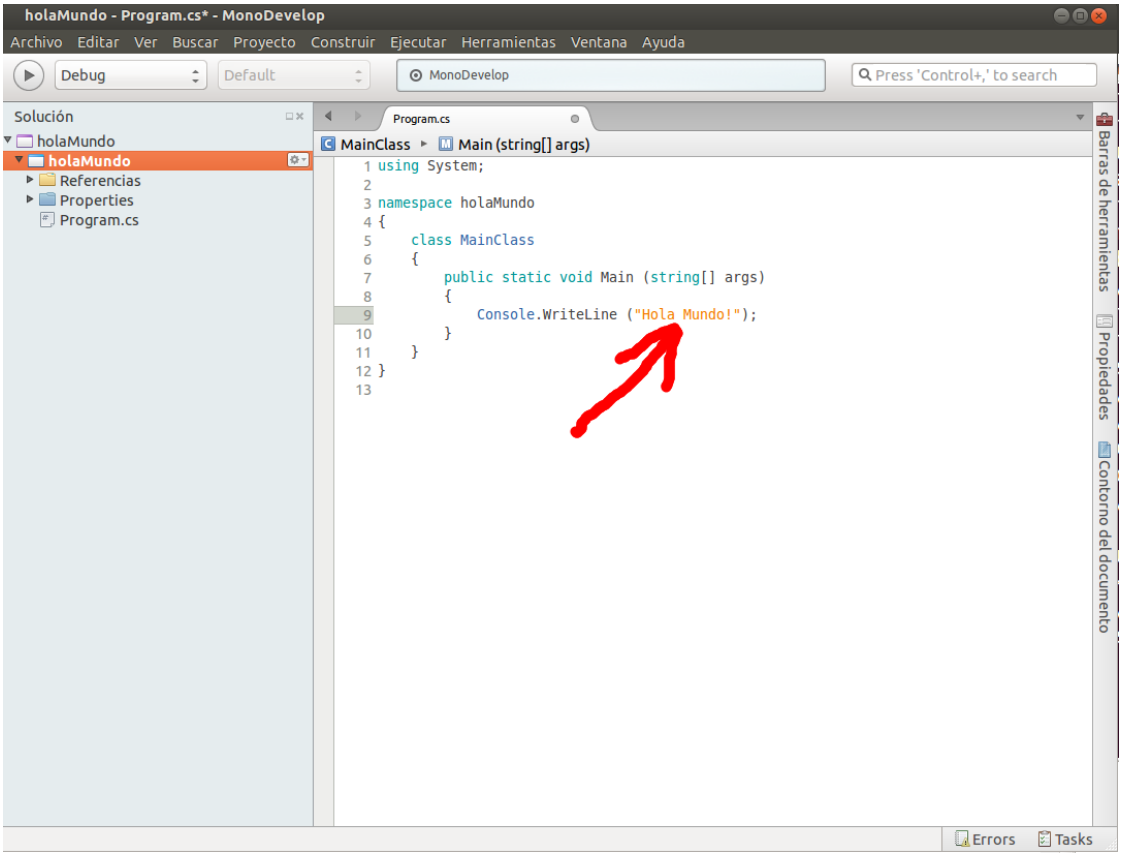
# El entorno MonoDevelop para C#

El entorno genera automáticamente una plantilla por defecto para nuestro programa:



# El entorno MonoDevelop para C#

Escribimos nuestro programa: en este caso es una modificación trivial de la plantilla.




# El entorno MonoDevelop para C#

Damos al play!

- El entorno lanza el compilador de C#
- Produce un ejecutable
- Y lo ejecuta