



# Programa-Me 2011

## Cómo Compilar y Ejecutar Programas en MAX 6.0

Patrocinado por



Realizado en



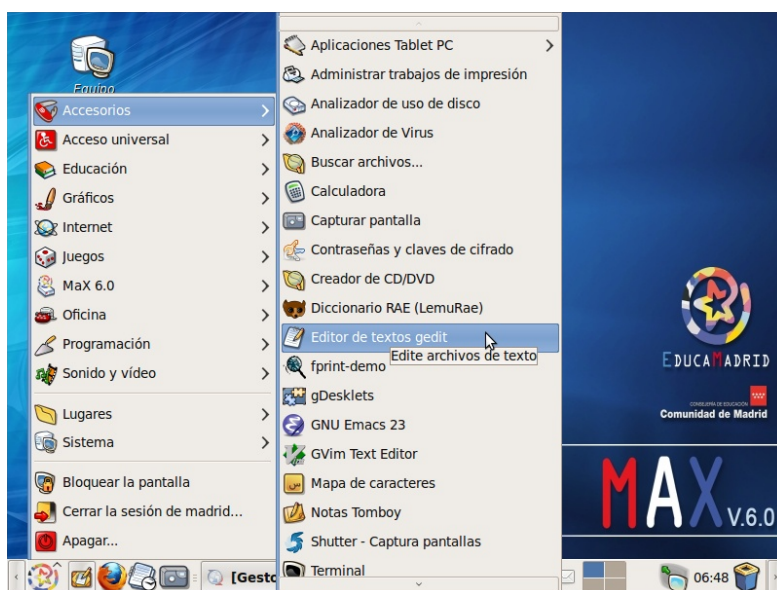
IES Antonio de Nebrija. Móstoles

# Cómo Compilar y Ejecutar Programas en MAX 6.0

## 1 ¿Por dónde empezar?

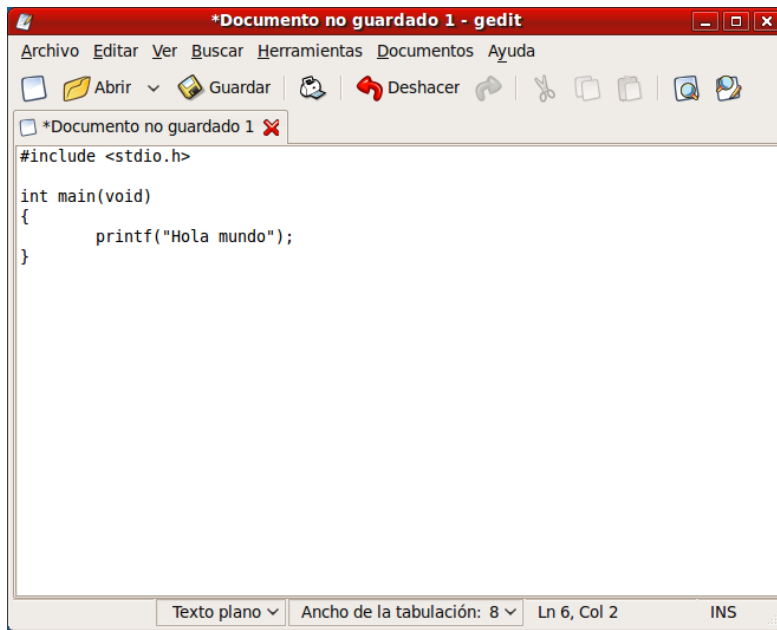
Una vez iniciamos Linux MAX 6.0, lo primero es abrir un editor de texto. Si no se tiene preferencia por alguno de los editores predeterminados de Linux, o no se ha utilizado ninguno, quizá lo más sencillo sea utilizar `gedit`.

Suponemos que, si se va a utilizar Vim-gnome o emacs es porque estamos familiarizados con el entorno Linux y no tendremos problema en abrir el editor y escribir el programa. De no ser así, una de las maneras de abrir un archivo de texto en MAX es la siguiente:



Cómo se ve en la figura anterior, accedemos al menú de inicio y pulsamos Accesorios — gedit. Esto nos abrirá un archivo de texto con dicho editor.

Una vez abierto el editor, no tengo más que escribir el programa:



The screenshot shows the gedit text editor window titled "\*Documento no guardado 1 - gedit". The menu bar includes Archivo, Editar, Ver, Buscar, Herramientas, Documentos, and Ayuda. The toolbar contains icons for Abrir, Guardar, Deshacer, and other editing functions. The main text area contains the following C code:

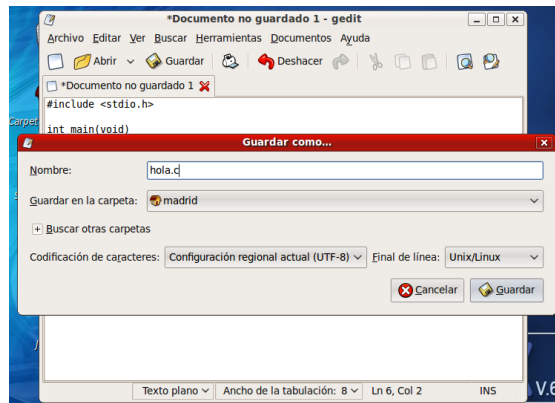
```
#include <stdio.h>

int main(void)
{
    printf("Hola mundo");
}
```

The status bar at the bottom indicates "Texto plano", "Ancho de la tabulación: 8", "Ln 6, Col 2", and "INS".

Cómo en cualquier otro editor, una vez escrito el programa, tendré que guardarlo con la extensión determinada para cada tipo de programa que quiera crear.

Por ejemplo, mi programa está escrito en C, luego la extensión con la que lo guardaré será `hola.c`

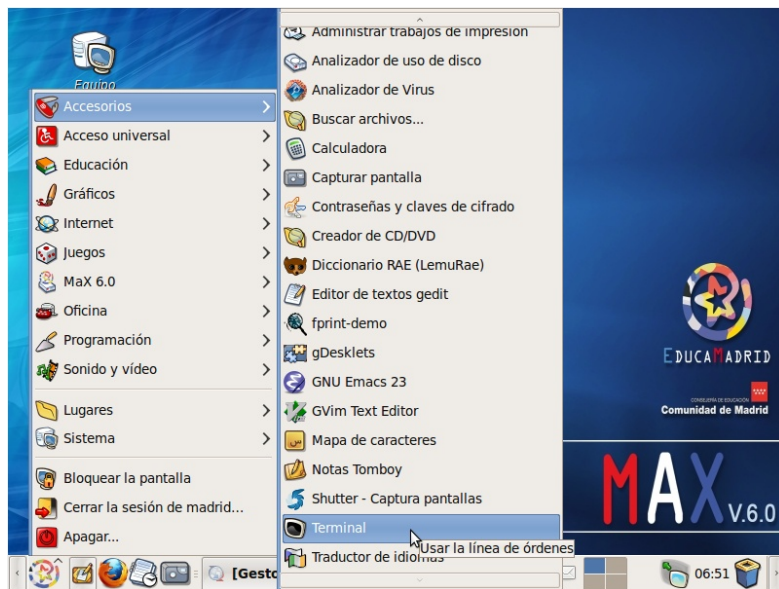


Según el tipo de programa realizado se guardará con la extensión `.c`, `.cpp` o `.java`.

## 2 Ya tengo mi programa escrito ¿Y ahora qué?

Una vez tengamos escrito nuestro programa, es hora de compilarlo.

Para ello abriremos una consola. Pero ¿cómo se abre una consola en MAX?



Esto nos abrirá una pantalla cómo la siguiente:



Una vez abierta, y situados en el directorio donde se encuentre nuestro archivo, escribiremos los comandos necesarios para compilarlo:

- Si el programa lo hemos escrito en C:

```
gcc <archivo.c>
```

*Ejemplo 1:* compilar el programa `hola.c`, y generar el fichero ejecutable `a.out`:

```
gcc hola.c
```

*Ejemplo 2:* compilar el programa `hola.c`, y generar el fichero ejecutable `hola`:

```
gcc hola.c -o hola
```

- Si el programa lo hemos escrito en C++:

```
g++ <archivo.cpp>
```

*Ejemplo 3:* compilar el programa `hola.cpp` y generar el ejecutable `hola`:

```
g++ hola.cpp -o hola
```

- Si el programa lo hemos escrito en Java:

```
javac <archivo.java>
```

*Ejemplo 4:* compilar el programa (la clase) `Hola.java` y generar el fichero (ejecutable a través de la máquina virtual de Java) `Hola.class`:

```
javac Hola.java
```

Al compilar nuestro archivo pueden pasar dos cosas: que el programa estuviera correctamente escrito y en este caso ya podríamos ejecutarlo, o que nos aparezcan errores de compilación. Entonces habría que regresar a nuestro código y modificar los posibles errores.

### 3 Mi programa ya compila, ¿qué es lo siguiente?

Ha llegado la hora de ejecutar el programa.

Si el programa está realizado en C o C++, sólo hay que escribir `./a.out` (o `./prog`, si al compilar hemos cambiado el nombre del ejecutable).

Si el programa esta realizado en Java, habrá que escribir `java programa` (dónde programa es el nombre que hemos dado a nuestro programa).

A partir de ahora sólo tendremos que introducir los valores que queramos para saber si la salida es la esperada.

### 4 ¿Cómo depuro mi programa?

Aunque en los equipos del concurso estará instalado `gdb`, la depuración está fuera del alcance de este documento. Si no se conoce su uso, es preferible que los participantes se acostumbren a depurar el código mediante la inclusión de líneas de redirección a pantalla con el fin de ir comprobando que es lo que hace el programa.

## 5 ¿Para qué sirven los archivos `sample.in` y `sample.out`?

Estos archivos son los ejemplos de prueba que aparecen en el enunciado del problema. Sirve para realizar algunas pruebas y así comprobar si el programa funciona correctamente. El archivo `sample.in` contiene el ejemplo de entrada y `sample.out` la salida esperada para dicho ejemplo. En el archivo `sampleLinux.out` tenemos la salida esperada para la ejecución del ejemplo en Linux. En caso de estar realizando el programa en MAX 6.0 o en cualquier otra distribución de Linux, si se desea comparar con la orden `diff`, deberemos utilizar éste último.

Al ejecutar una orden en Linux, podemos dirigir el contenido de un archivo a la entrada estándar. De esta manera, las funciones que toman los valores de la entrada estándar cogerán dichos valores del fichero de entrada en lugar del teclado. Para ello basta con escribir

```
./prog < sample.in
```

donde `prog` es el nombre del programa que hayamos creado o

```
java prog < sample.in
```

Si queremos que el resultado se obtenga en un fichero, en lugar de en la salida estándar no tenemos más que redirigir la misma hacia otro fichero, por ejemplo `result.out`:

```
./prog < sample.in > result.out
```

Si queremos comprobar si el fichero que nosotros hemos obtenido (`result.out`), es igual al fichero correcto `sample.out` no tendremos más que utilizar la orden `diff` para saber si hay diferencias y, de haberlas, dónde se encuentran. Si no se obtiene salida, significará que ambos ficheros son iguales, y nuestro programa ha funcionado bien con los casos de ejemplo.

Deberemos tener en cuenta que eso no significa que la solución funcione correctamente para todas las posibles entradas. Los jueces utilizarán otros casos de prueba distintos para comprobar que el programa funciona bien. Es posible que, aún funcionando correctamente para los casos de ejemplo propuestos en el enunciado, falle con los casos de prueba utilizados por los jueces.