C LENGUAJE ESTRUCTURADO. EL COMPILADOR

3.1.- Origen del lenguaje C

El lenguaje de programación C fue desarrollado originalmente por Dennis Ritchie, en los Laboratorios de Bell Telephone en 1972 como resultado de dos lenguajes anteriores que se desarrollaron también en dichos Laboratorios, el B y el BCPL. El lenguaje C estuvo confinado al uso exclusivo de los laboratorios Bell hasta 1978, añó en el que Brian Kernighan y Ritchie publicaron una descripción definitiva del lenguaje (*Brian W. Kernighan y Dennis M. Ritchie, The C Programming Language, Prentice-Hall, 1978*).

Tras la publicación de las definiciones de C, los profesionales de la Informática, impresionados por las muchas características deseables del C , comenzaron a promover el uso del lenguaje. A mediados de los años ochenta la popularidad de C se había extendido por todas partes. Se habían escrito numerosos compiladores e intérpretes para C para ordenadores de todos los tamaños y se habían desarrollado numerosas aplicaciones comerciales. Es más, muchas aplicaciones que se habían escrito originariamente en otros lenguajes, se reescribieron en C, dando así cuenta de su eficiencia y portabilidad.

Las primeras implementaciones comerciales de C diferían en parte de la definición original de Kernighan y Ritchie, creando pequeñas incompatibilidades entre las diferentes implementaciones del lenguaje. Estas diferencias disminuyen la portabilidad que el lenguaje intentaba proporcionar. Consecuentemente, El Instituto Nacional Americano de Estándares (ANSI) ha desarrollado una definición estandardizada del lenguaje C. La mayoría de los compiladores e intérpretes actuales de C adoptan el estándar ANSI aunque algunos compiladores pueden proporcionar características adicionales propias.

En la década de los ochenta, Bjarne Stroustrup (*The Cprogramminig Language, Segunda Edición, Addison - Wesley* ,1991) desarrolló, también en los laboratorios Bell otro lenguaje de programación de alto nivel denominado C++. Éste se basa en C y, por tanto, todas las características estándar de C están disponibles en C++. Sin embargo, C++ no es una mera extensión de C. Incorpora nuevos fundamentos que constituyen una base para la ***Programación Orientada a Objetos***, un nuevo paradigma de la programación de interés para los programadores profesionales.

C no puede ser considerado un lenguaje de alto nivel como ADA, Pascal, Basic, etc. ya que permite especificar y controlar detalles de implementación de una aplicación para conseguir la máxima ***eficiencia*** del ordenador. Por otro lado, no puede ser considerado un lenguaje de bajo nivel, como los lenguajes ensambladores, pues permite ocultar detalles de la arquitectura del ordenador, aumentando así la ***eficiencia*** en la programación. Por ello, el lenguaje C es normalmente considerado un lenguaje de medio nivel.

3.1.1.- Ventajas y desventajas de C

Algunas características del lenguaje C pueden ser consideradas puntos fuertes o ventajas del mismo, mientras que otras pueden ser consideradas como puntos débiles o desventajas:

Entre las ventajas de C destacan:

1ª.- Tener pocas reglas de sintaxis y un conjunto reducido de palabras clave o reservadas

2ª.- Los programas en C se ejecutan a velocidades próximas a la de los lenguajes ensambladores por la combinación de un lenguaje pequeño, un sistema de tiempo de ejecución reducido y el hecho de que el lenguaje se ciña al hardware.

3ª.- Es ***poco o débilmente tipado***, es decir, permite ver los datos como tipos distintos dependiendo de la fase o instrucción del programa en la que se encuentre.

4ª.- Es un lenguaje ***estructurado***, pues contiene todas las estructuras de control de los lenguaje estructurados (secuenciales, repetitivas, condicionales). Permite además la creación de bloques para reunir varias sentencias entre llaves que se comportan como un todo.

5ª.- Proporciona soporte para programación modular ya que es capaz de separar la compilación y el enlazado (linkado), permitiendo así la recompilación exclusivamente de aquellas partes del programa que han sido cambiadas durante el desarrollo del mismo.

6ª.- Se acopla fácilmente a las rutinas del lenguaje ensamblador, es decir, que desde un programa se puede llamar fácilmente a una rutina de ensamblador y viceversa.

7ª.- Permite la manipulación directa de bits mediante un conjunto amplio de operadores específicos para ello.

8ª.- Es ***portable***, pues los programas C escritos sobre una máquina o sistema operativo concreto se pueden transferir fácilmente a otros.

9ª.- Proporciona una ***biblioteca*** extensa de funciones especiales, en la que están incluidas la mayoría de operaciones y funciones que se utilizan habitualmente en el desarrollo de un programa C.

10ª.- Permite la manipulación de estructuras de datos ***dinámicas*** mediante punteros y técnicas de memoria dinámica.

Por su parte, los principales puntos débiles o desventajas de C están en:

1ª.- La ausencia de tipos estrictos, lo que permite que a una variable de tipo determinado se le asigne un valor de otro tipo (por ejemplo, a una variable de tipo char se le puede asignar un valor de tipo int).

2ª.- La ausencia de verificación en tiempo de ejecución. Así, por ejemplo, no se controla ni se advierte, en tiempo de ejecución, que se han sobrepasado los límites de un vector.

3ª.- Es un lenguaje de gran dificultad de comprensión y aprendizaje.

3.2.- Estructura de un programa en C

Todo programa en C consta de uno o mas módulos llamados *funciones*. Una de las funciones se llama siempre ***main.***

Los programas escritos en lenguaje C se almacenan en ficheros de texto estándar, denominados ***ficheros fuente***, cuya extensión es .c o .cpp con la siguiente estructura:

A.- **Cabecera: Comentario inicial y entorno**

Consiste en un texto introducido por medio de una cadena de caracteres y delimitado por los símbolos /* y */, que indican al compilador que su contenido no debe ser tratado como una instrucción. Este texto contiene alguna aclaración o una explicación general de lo que hace el propio programa.

La cabecera además contiene las *directivas para el preprocesador* que permiten definir *macros* e incluir el contenido de otros archivos durante la compilación. Las mas habituales son las que permiten definir constantes (***#define***) y las que permiten utilizar funciones de librería (***#include***).

La cabecera es también el lugar idóneo para definir, siempre quie se considere necesario, las *constantes y las variables globales del programa* y donde deben declararse los *prototipos de las distintas funciones* que se van a utilizar

B.- **Función Principal**

Todo programa C debe contener una función nombrada "***main***" que representa la función principal del programa, por donde éste comienza a ejecutarse

La función main normalmente se define como un procedimiento, es decir, no devolviendo ningún dato (aunque puede no hacerlo), el tipo ***void*** indica precisamente que la función no devuelve un tipo concreto de datos.

El cuerpo de la función se inserta entre llaves { . . . }

```
Void main ( )
{
        Sentencias
}
```

Una ***sentencia o instrucción*** es la unidad ejecutable mas pequeña en un programa C. Las sentencias controlan el flujo u orden de ejecución. Una sentencia C puede formarse a partir de una palabra clave (**for, while, if . . . else**, etc.), expresiones, declaraciones o llamadas a funciones. Cuando se escribe una sentencia hay que tener en cuenta las siguientes consideraciones:

- Toda sentencia simple termina con un punto y coma (;)
- Dos o mas sentencias pueden aparecer sobre la misma línea, separadas una de otra por un punto y coma, aunque esta forma de proceder no es aconsejable pues va en contra de la claridad que se necesita cuando se lee el código de un programa.

- Una sentencia nula consta solamente de un punto y coma. El uso de sentencias nulas se circunscribe casi únicamente a la utilización de la sentencia while.

- Una ***sentencia compuesta o bloque*** es una colección de sentencias incluidas entre llaves ({ }). Un bloque puede contener a otros bloques.

C.- **Definición o implementación de las funciones que se han declarado en el entorno**

La declaración de una función es de la forma:

Tipo_RESULTADO NOMBRE_FUNCION (Parámetros formales)

{

Sentencias

}

Una función se ***declara*** en el entorno de la cabecera y se ***desarrolla*** en cualquier lugar del programa, normalmente después de la función principal. Cuando ésta la llama para hacer uso de ella, le pasa los ***valores actuales*** para que opere sobre ellos. Estos parámetros (salvo en el caso de las matrices) se pasan ***por valor,*** esto es, sustituyen a los parámetros formales que aparecen en la definición de la función, pero la función opera sobre una copia de los valores reales, no sobre la dirección de memoria en la que se almacenan. Por tanto, no se alteran los valores actuales sino que se respetan estos valores que poseen las variables antes de ser utilizados por la función.

Si se desea que los valores actuales sean modificados por la función deberá realizarse una llamada a la función ***por referencia***

El programa siempre comenzará por la ejecución de la función ***main***, la cual puede acceder a las demás funciones. Las definiciones de las funciones adicionales se deben realizar aparte, bien precediendo o bien siguiendo a ***main***.

Cada función debe contener:

1º.- Una ***cabecera*** de la función, que consta del nombre de la función, seguido de la lista opcional de los ***argumentos*** encerrados entre paréntesis

2º.- Una lista de ***declaración*** de argumentos, si se incluyen éstos en la cabecera.

3º.- Una ***instrucción compuesta*** que contiene el resto de la función.

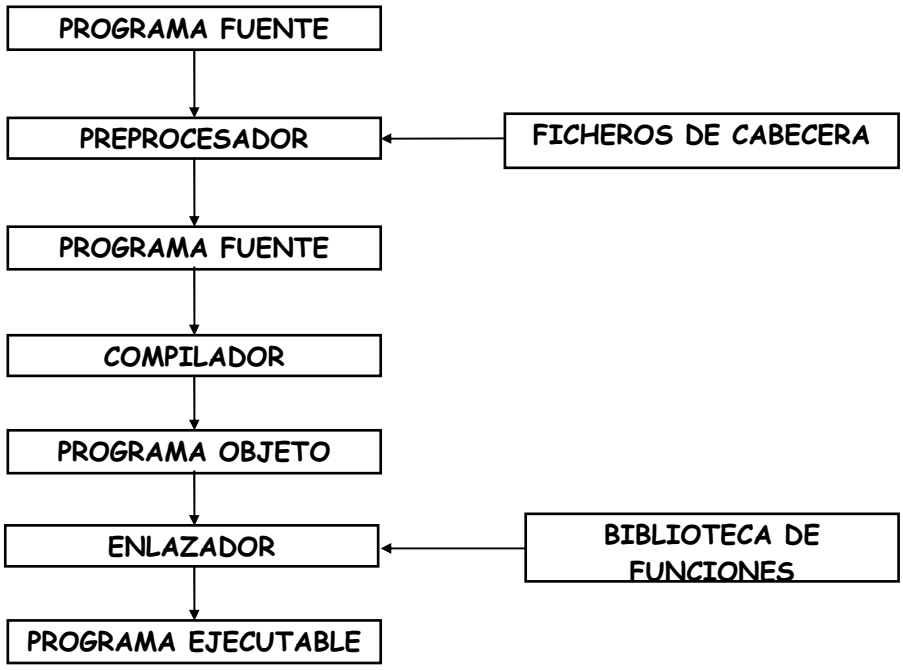
Los argumentos de la función son símbolos que representan información que se le pasa a la función desde otra parte del programa (normalmente se llama ***parámetros*** a los argumentos.

Cada instrucción compuesta se cierra con un ***par de llaves,*** **{ }**. Las llaves pueden contener combinaciones de instrucciones elementales (denominadas ***instrucciones de expresión***) y otras instrucciones compuestas. Así las instrucciones compuestas pueden estar anidadas, una dentro de otra. Cada instrucción debe terminar en un punto y coma (**;**).

Los ***comentarios*** pueden aparecer en cualquier parte del programa, mientras estén situados entre los delimitadores **/*** y ***/** (por ejemplo: ***/* esto es un comentario */***). Los comentarios son útiles para identificar los elementos principales de un programa o para explicar la lógica subyacente en éstos.

3.3.- Herramientas para la elaboración y la depuración

En general, el proceso seguido para obtener un programa ejecutable desde un programa fuente escrito en C es el indicado en la figura.



Para realizar estos cometidos se utilizan distintos paquetes, mas o menos integrados. Estos paquetes contienen todos los archivos ejecutables y las librerías que se necesitan para crear, compilar, enlazar, depurar y ejecutar un programa escrito en C.

Los programas pueden compilarse de dos formas distintas: **Mediante un entorno integrado de desarrollo**, que engloba el editor de textos para escribir el programa fuente, el compilador y el enlazador pudiéndose realizar todas las operaciones sin necesidad de salir del entorno. O bien mediante el **compilador en línea de comandos** que permite compilar y enlazar separadamente el programa fuente, que debe estar escrito mediante un editor de textos sin formato.

Entre los paquetes integrados mas utilizados se encuentran: Turbo C y turbo C++ de Borland, C++ de Borland, QuickC y C compiler de Microsoft etc.

Por su parte, de cara a localizar errores en ficheros ejecutables existen depuradores específicos como el Code View de Microsoft o el gdb para UNIX. Ambos disponen de opciones de rastreo, punto de parada, ventanas de seguimiento ejecución controlada y ejecución automática.

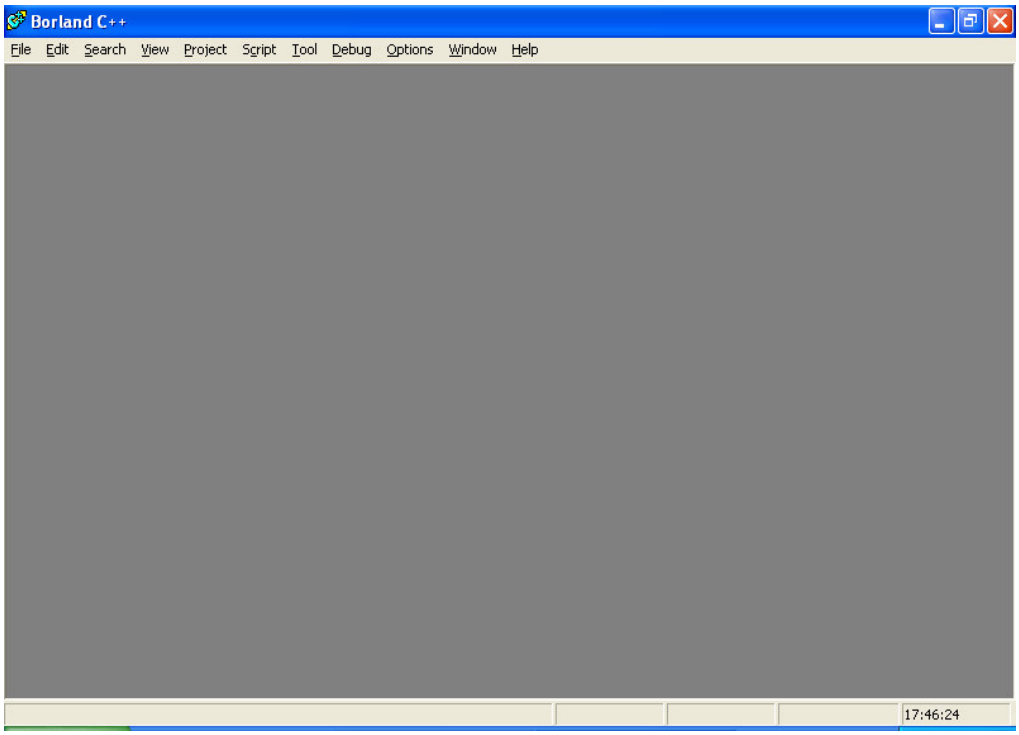
3.4.- El compilador Borland C++

El compilador Borland C++ es un compilador profesional optimizado que contiene en realidad dos compiladores:

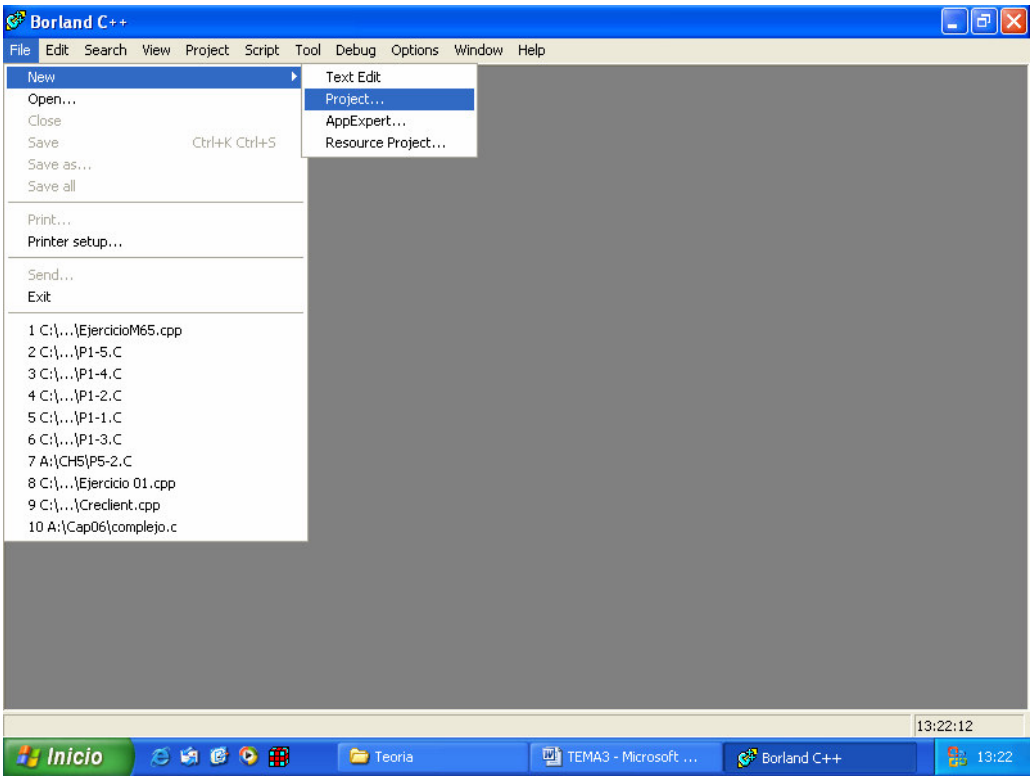
- Un compilador para lenguaje C estándar (ANSI C) que traduce los archivos de texto de extensión .c
- Un compilador C++ (que cumple el estándar AT&T) que traduce los archivos de texto de extensión .cpp

Aunque se puede utilizar indistintamente cualquier compilador y la elección de uno u otro se hace de forma automática, es conveniente utilizar desde el principio la extensión .cpp, ya que así se trabajará siempre con el compilador C++ con una ligera ventaja sobre el compilador C: *el compilador C++ señala errores en lugares donde C no ofrece mas que una advertencia*.

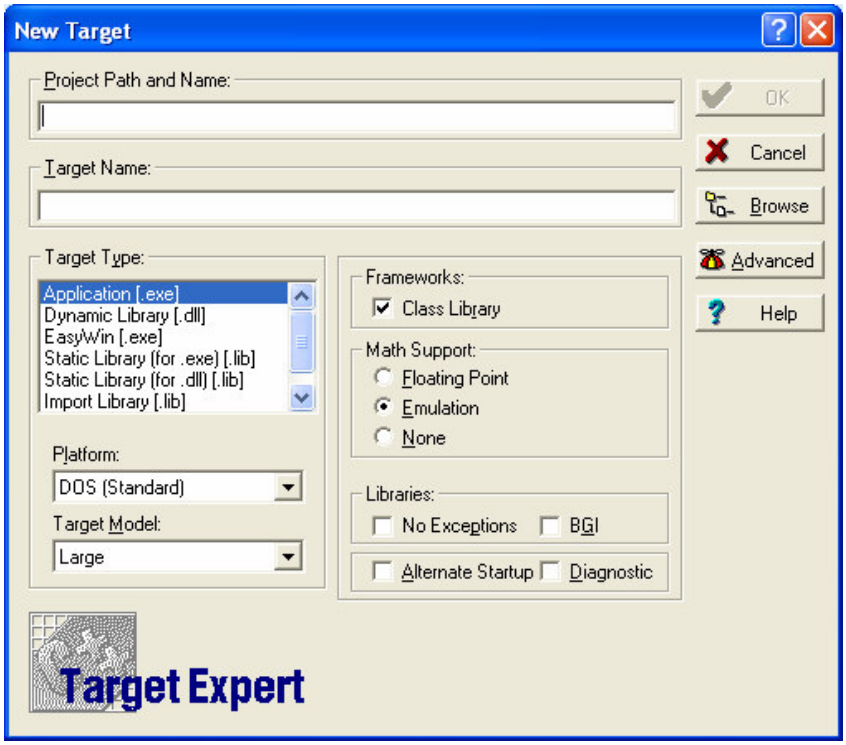
La barra de menús es la forma de acceso principal a todas las órdenes de menús. La estructura general del menú principal será la de la figura:



Sobre este menú principal, activando el submenú ***file*** (archivo) puede seleccionarse un ***new Project*** (nuevo proyecto):

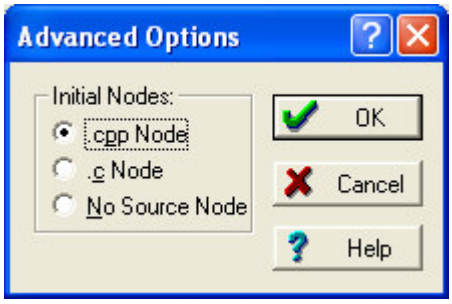


Apareciendo el siguiente menú de configuración del proyecto para una plataforma determinada, en concreto ***DOS estándar***:



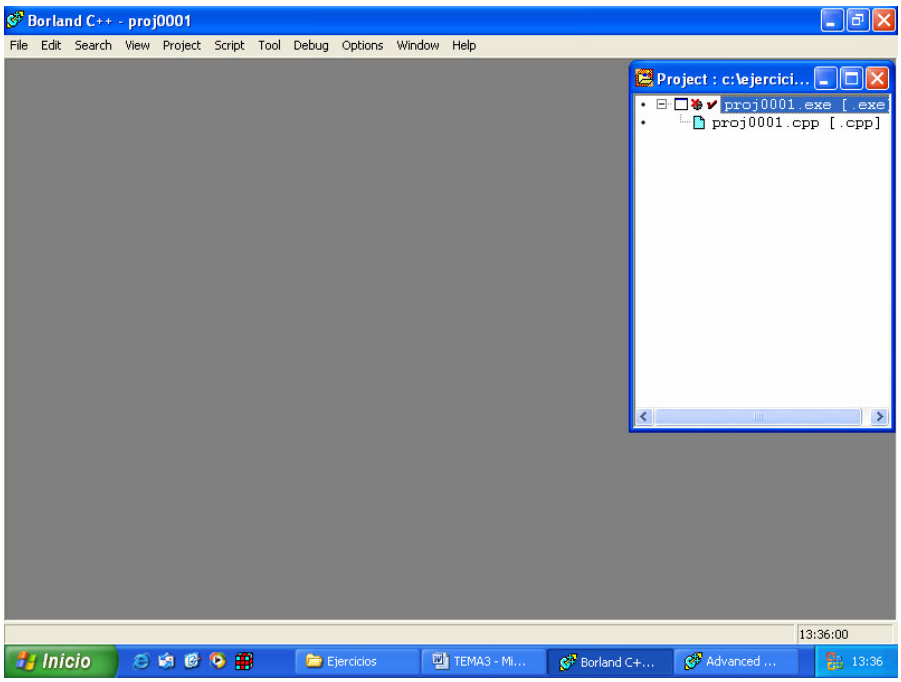
En el que se solicita el *nombre del proyecto y la carpeta* en la que será incluido. El proyecto creado tendrá extensión **.ide** Por ejemplo c:\Ejercicios\Ejer1.ide.

El pulsar el botón de opciones avanzadas ***(Advanced)*** se obtiene una pantalla como la de la figura:

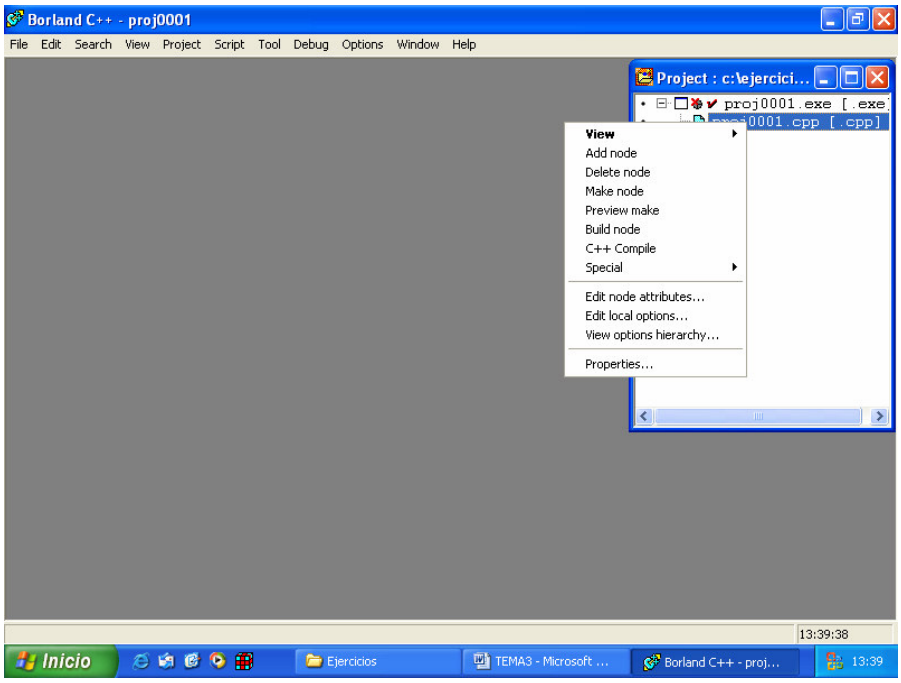


En la que se indica el compilador que se va a utilizar, que, como se dijo anteriormente el valor de omisión es el compilador C++

Al validar tanto las opciones avanzadas como la creación del proyecto, éste se inicializa creando el primer fichero fuente de extensión .cpp:

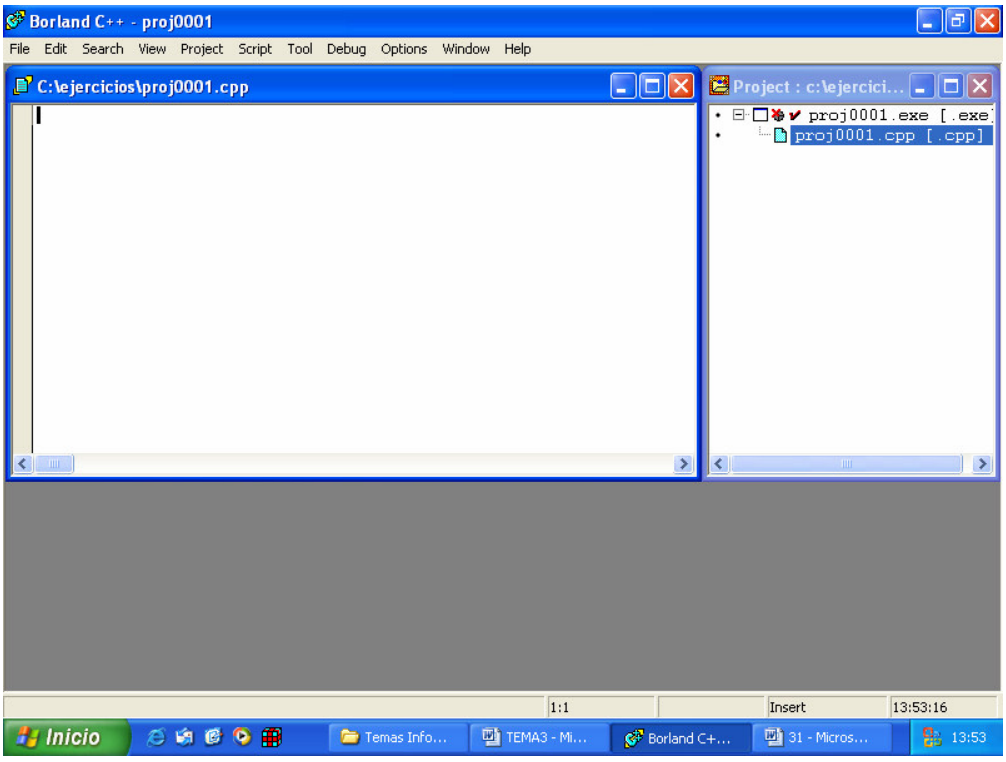


Al que el sistema le asocia un nombre de omisión (en este caso proj0001.cpp). Si se desea alterar este nombre, mediante el menú contextual de dicho fichero (activando el botón derecho del ratón al tener el fichero el foco):

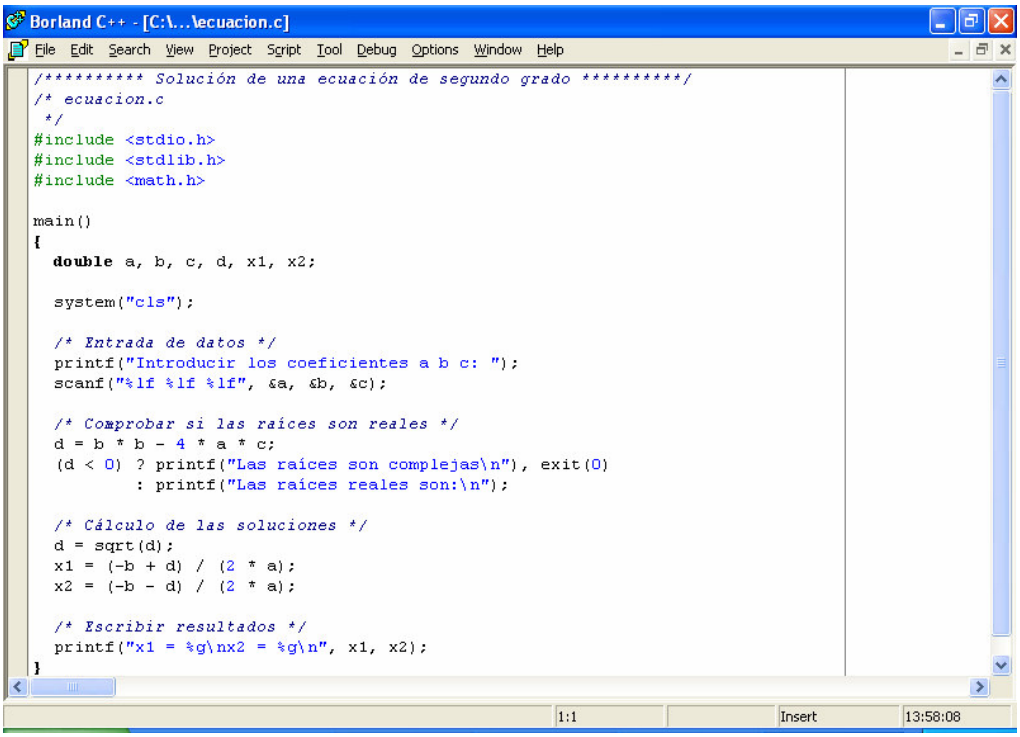


Puede eliminarse seleccionando la opción ***Delete Node*** (Eliminar Nodo) y bien puede crearse un nuevo fichero fuente mediante ***Add Node*** (adicionar nodo)

Una vez creado el fichero fuente, se abre pulsando sobre él dos veces con el ratón, quedando de una apariencia parecida a la de la figura:



Una vez introducido el código fuente:

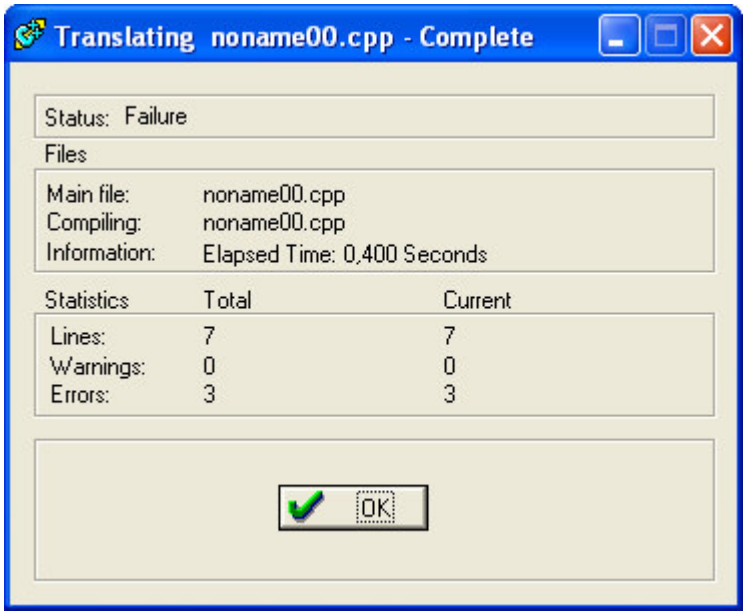


Puede ***compilarse el programa*** mediante la opción ***compile*** del menú ***Project*** que permite detectar los posibles errores.

Si no existen, se creará un ***programa objeto*** y podrá ejecutarse el progre desde el menú principal ***debug*** apartado ***run.***

Si existiesen errores en el programa fuente el compilador avisaría de ello indicando cuántos y dónde podrían aparecer dichos errores, abortando posteriormente la operación de compilación.

La apariencia de la detección de errores será la siguiente:



Una vez corregidos los errores el programa podrá ser ejecutado.