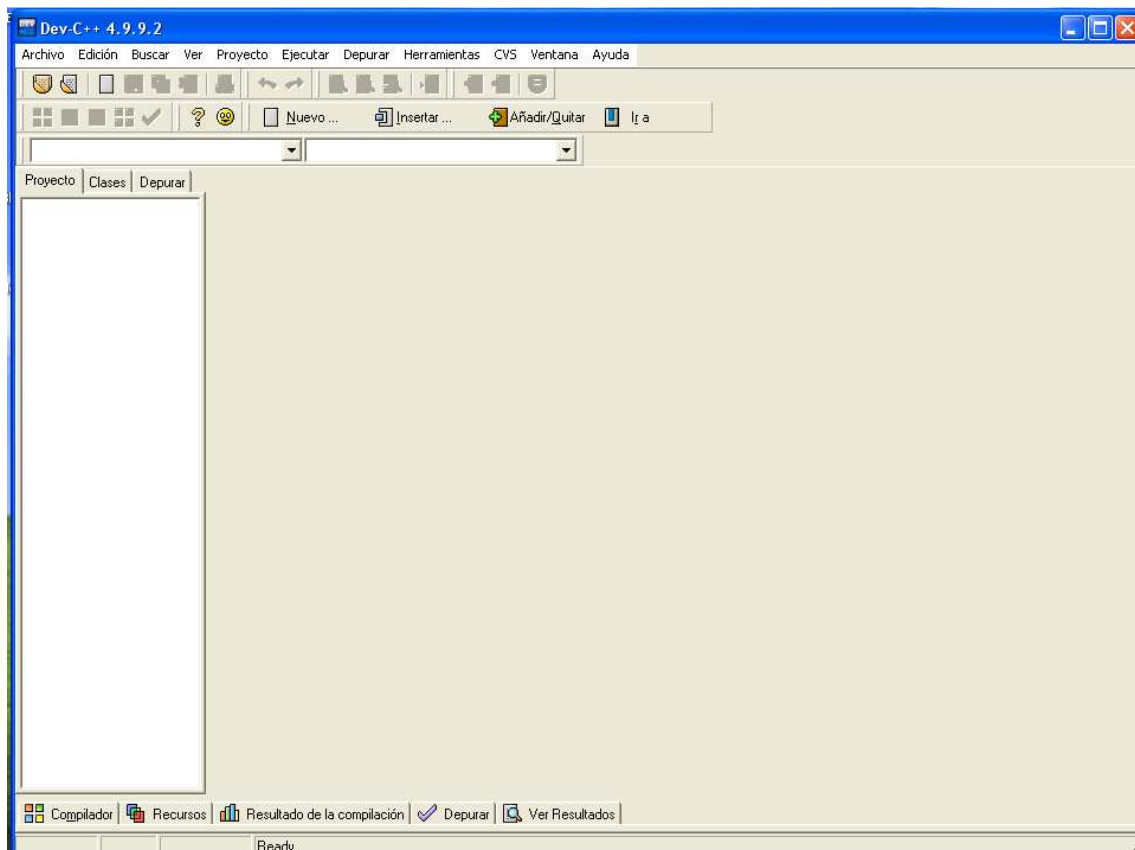


Trabajando con DEVC

PRACTICA 1

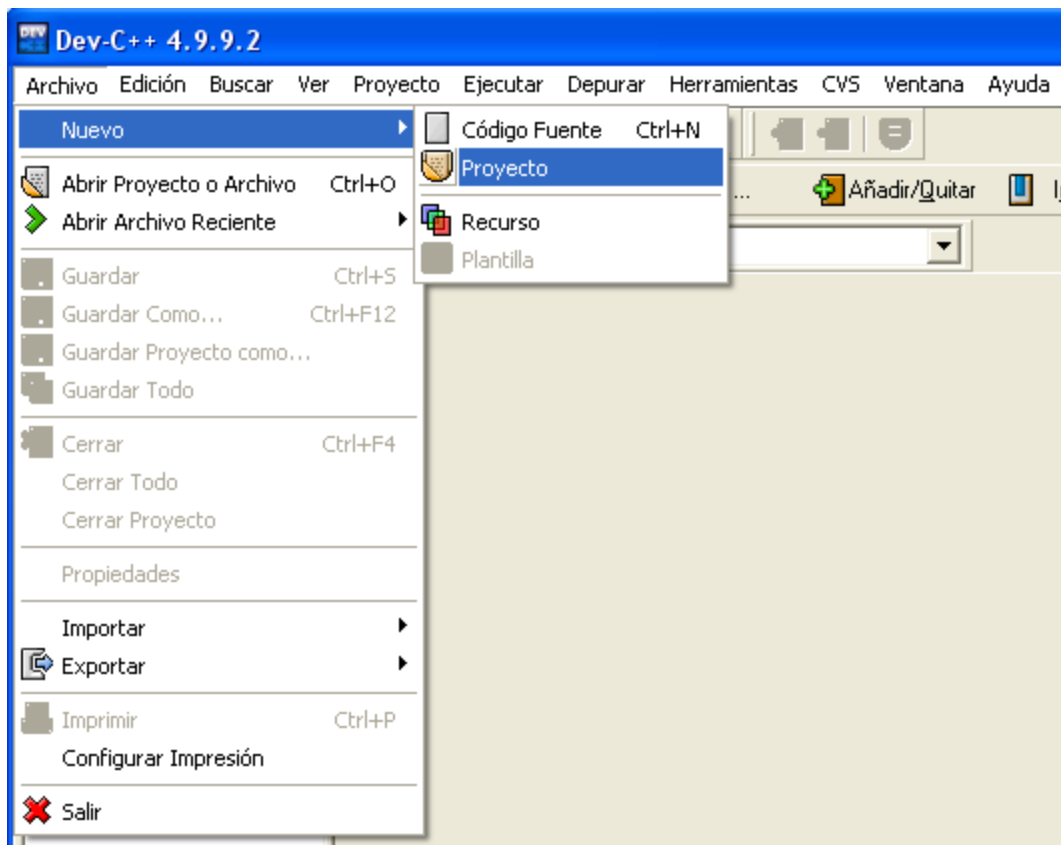
PARTE 1

Después de Instalar, escoger idioma y que trabaja con c; ya aparece nuestro compilador listo para trabajar

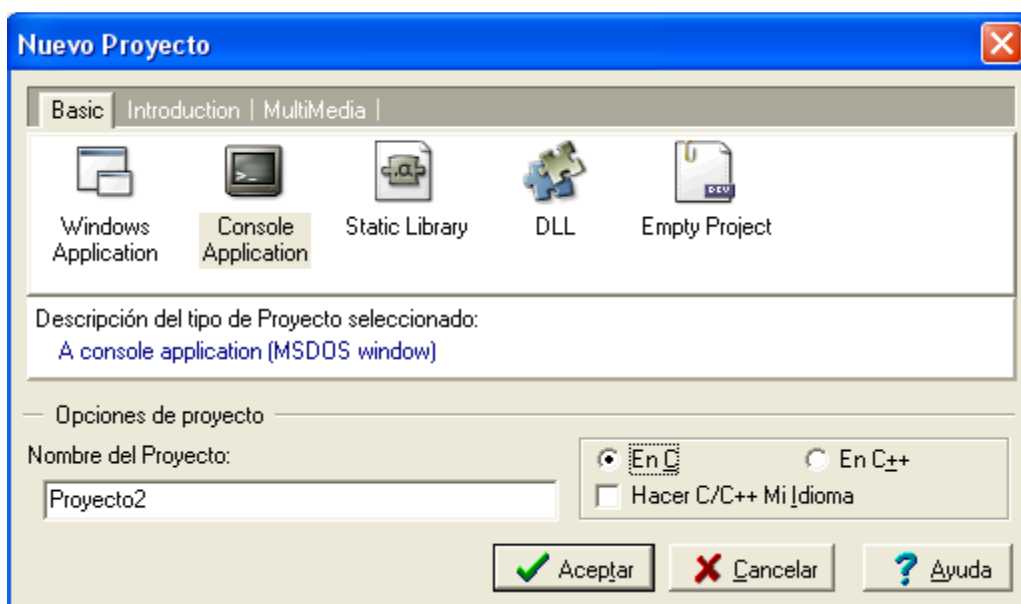


Debemos crear un proyecto SIEMPRE donde este contenido nuestro programa principal (main).

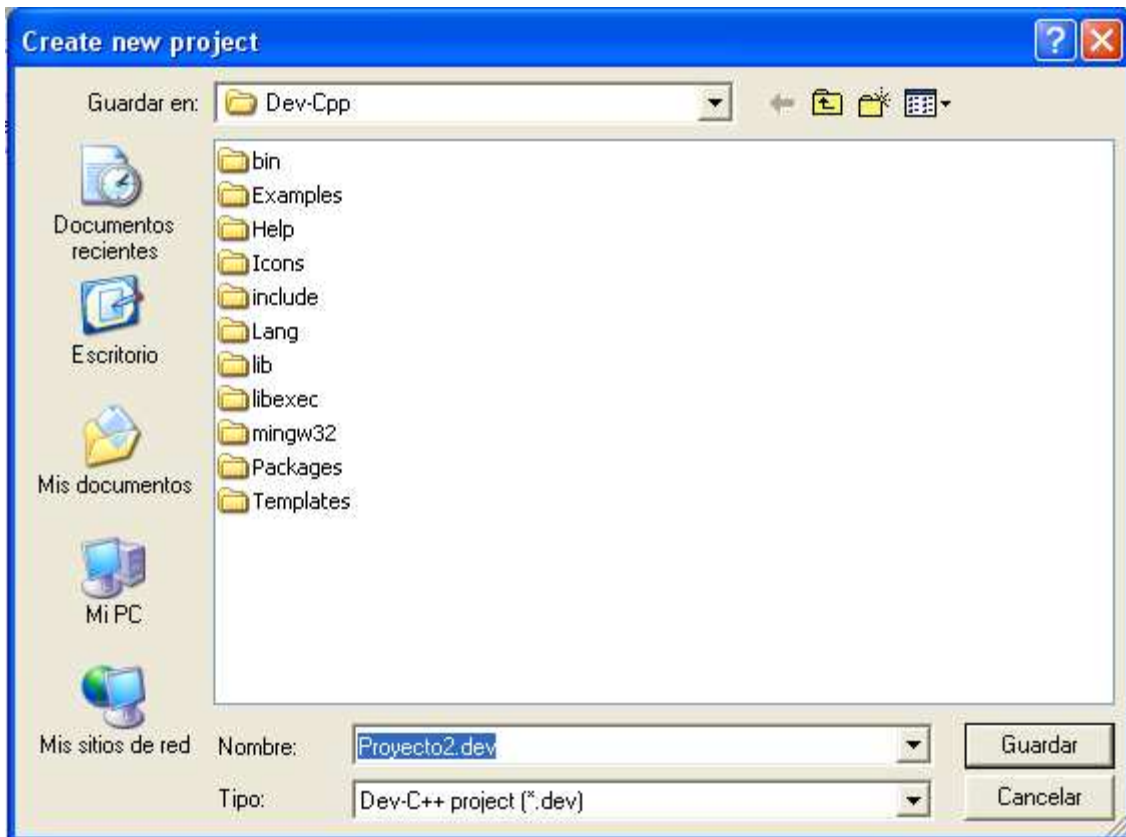
1. Seleccione del menú “archivo” la opción “nuevo”, y en el menú desplegable seleccione: “proyecto”.



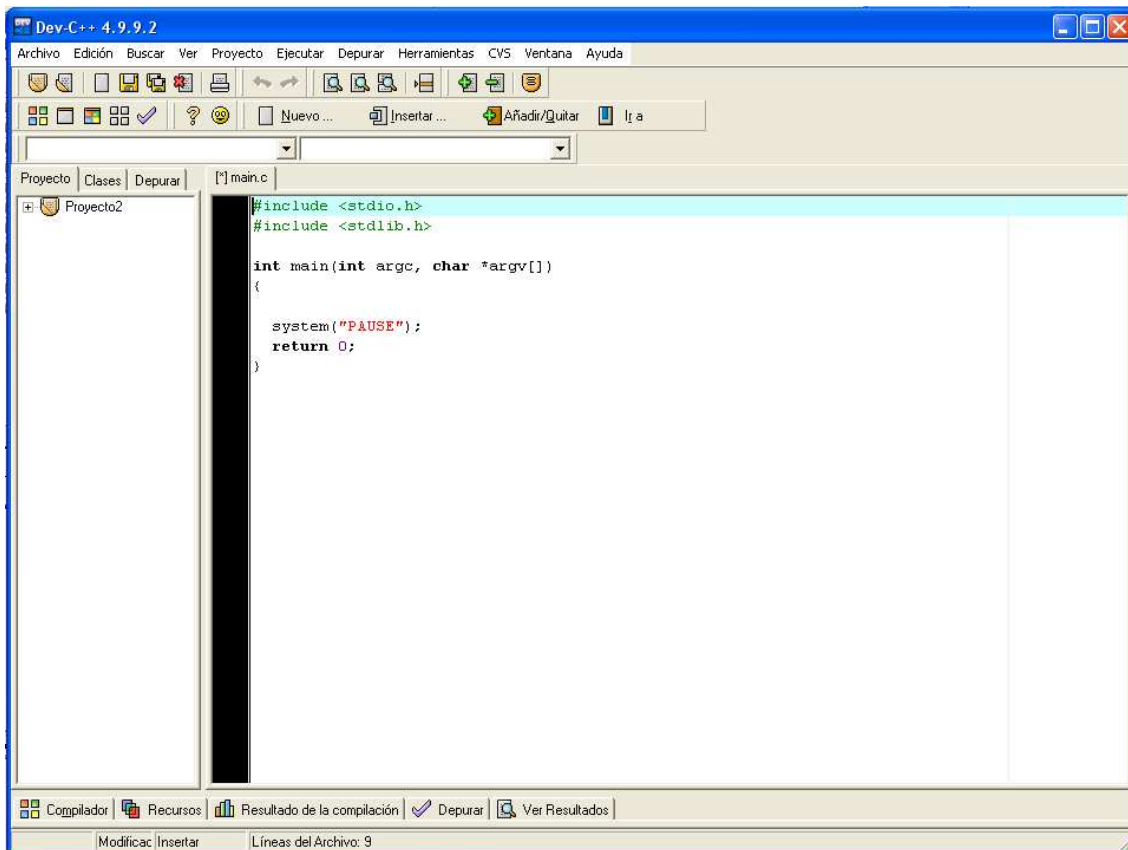
2. Aparece caja de Dialogo donde debe escogerse el proyecto nuevo a trabajar; escogeremos Aplicaciones de consola (Console Application), para trabajar en C, y el nombre de proyecto sera Proyecto2 (Puedes ponerle el nombre que quieras).



3. Confirma guardar para que se cree el nuevo proyecto.



Aparece por default esta pantalla. Lista para solo mostrar la consola de aplicación (negra) al ejecutar el programa.



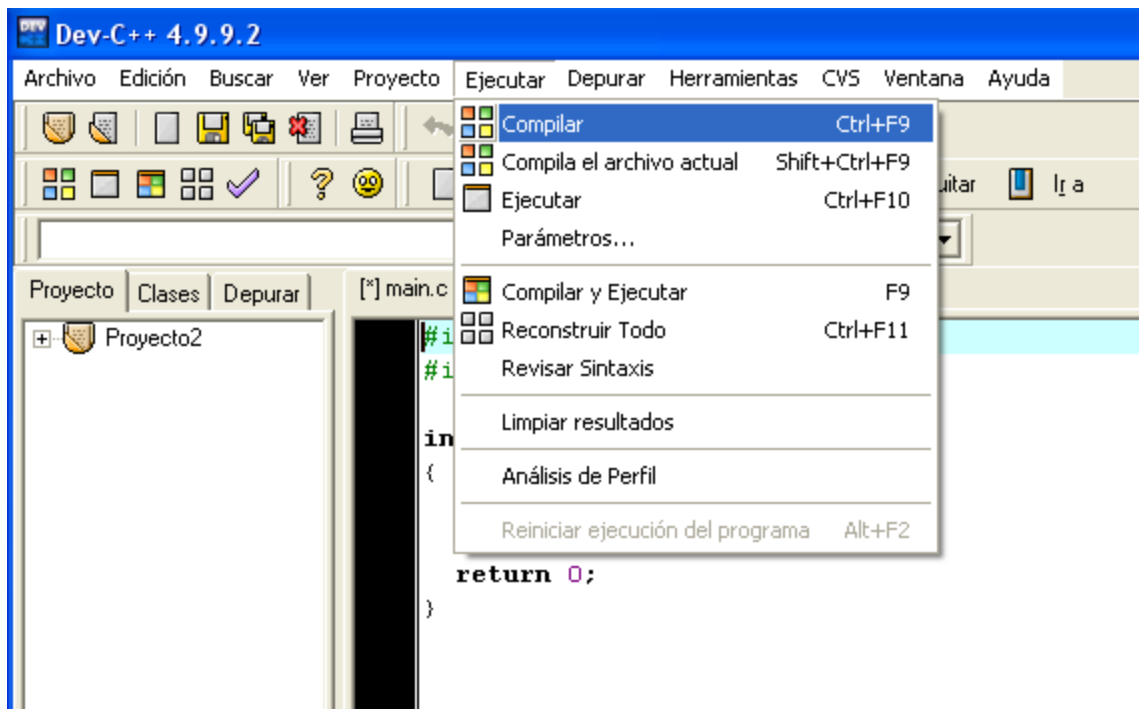
Ejemplo muestra, más cerca para observar que contiene el programa.

```
#include <stdio.h>
#include <stdlib.h>

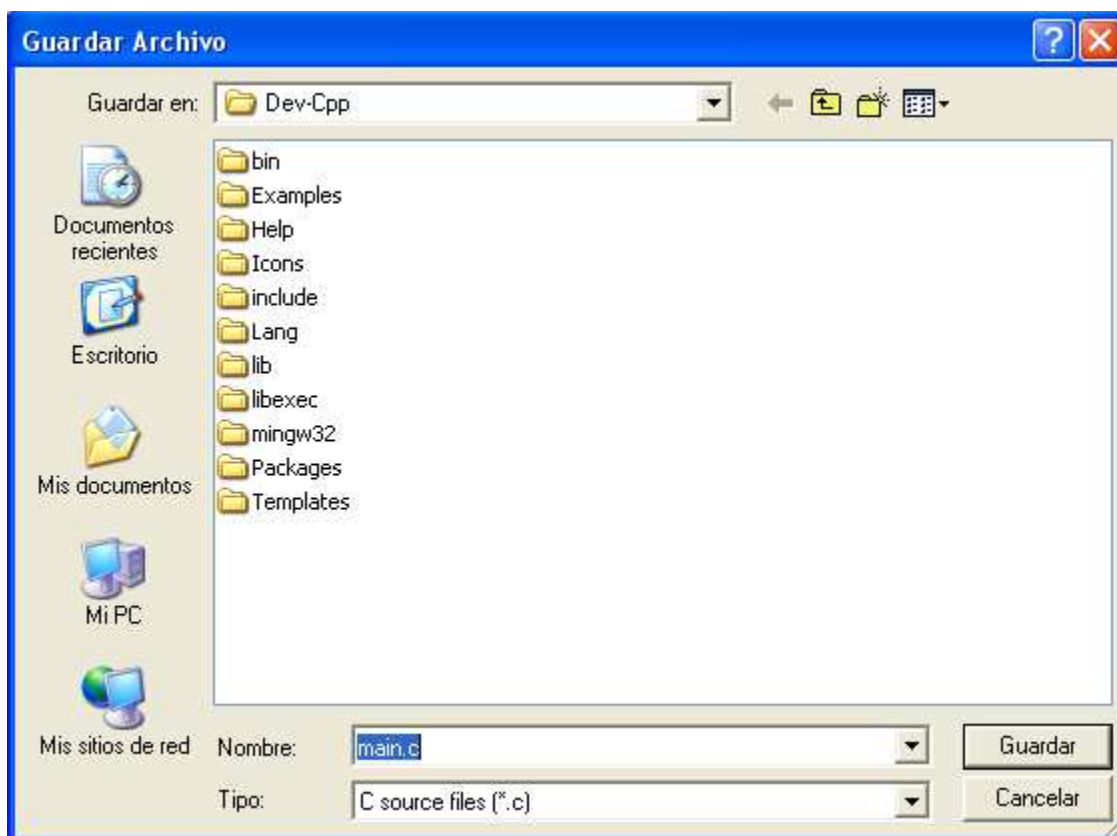
int main(int argc, char *argv[])
{
    system("PAUSE");
    return 0;
}
```

Compile y corra un programa

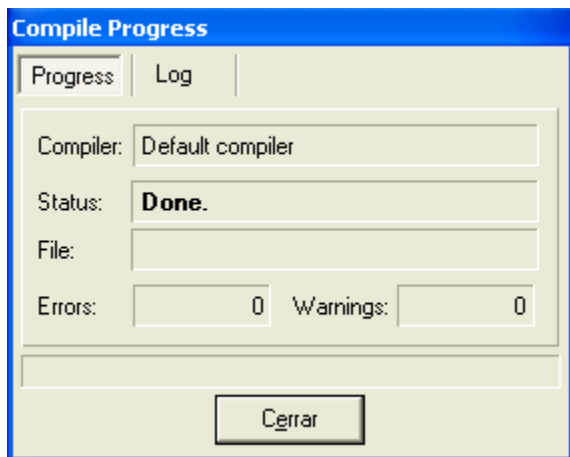
4. Para realizar estas dos acciones escoger menú "ejecutar", y después seleccionar "compilar". Con este recurso se pasa a unos y ceros (si el programa no tiene errores), antes te dice si la estructura, letras minúsculas, mayúsculas y comandos están escrito como se pide.



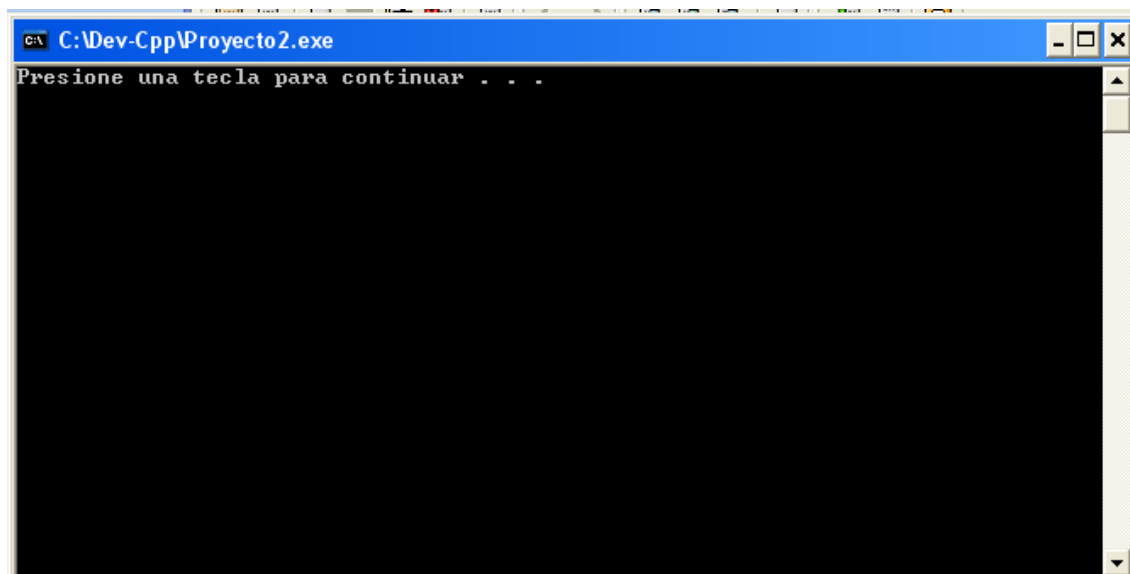
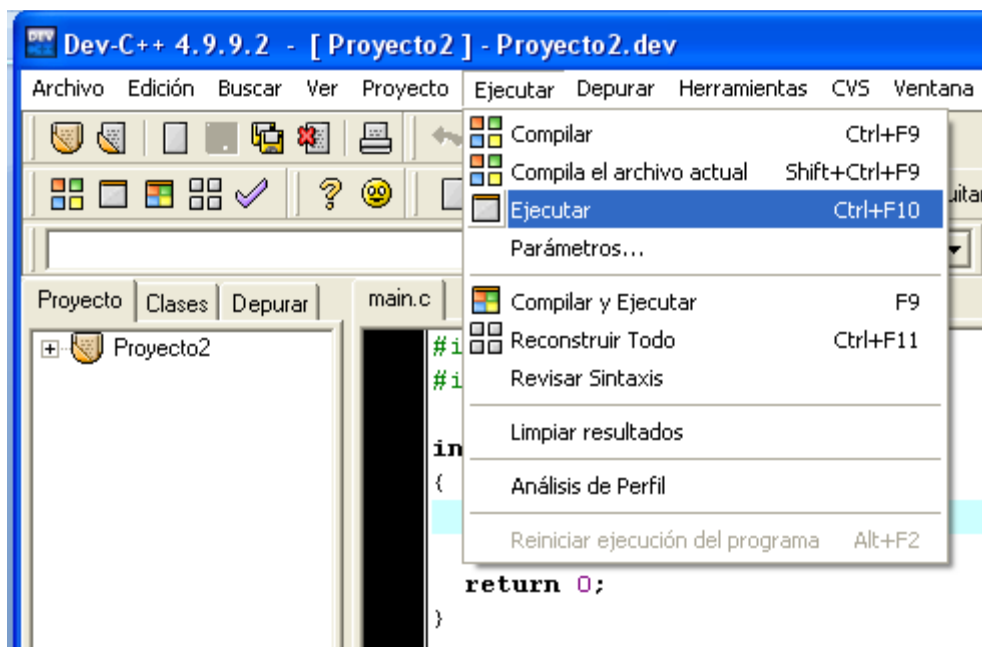
5. Antes de compilar debes guardar, si se te olvida, el software te avisa que salves tu programa principal (main) con un nombre cualquiera mas el punto y la letra C.



Aparece caja de dialogo del progreso de compilación. Y si no hay error aparece CERO errores.



Si no hay errores puedes Ejecutar o Correr el programa. En el menú "Ejecutar", escoger submenú "Ejecutar" (otra vez). Saldrá una pantalla negra con resultado.



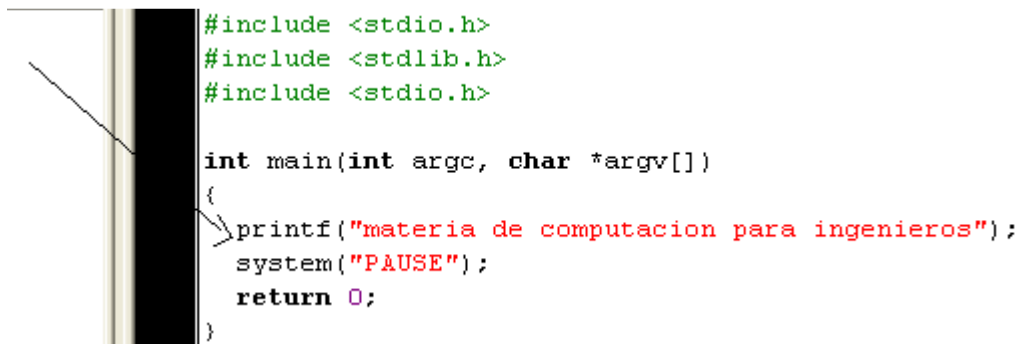
Parte 2. Ejecuta un comando que imprima un letrero

Escribe una línea de código: `printf("materia de computación para ingenieros");`

Nota: al final de este comando que muestra letreros lleva un punto y coma; es tan importante escribirlo que más tarde se verá que pasaría si omites este.

1. Primero tenemos la captura de pantalla de cuando escribimos nuestra primera orden.

Dependiendo el compilador es la asignación de cabeceras `#include <>`, por ejemplo `printf` utilizar la cabecera `stdio.h`, aunque solo en algunos compiladores, el más común es Borland, pero en DEVC puede omitirse, si se escribe no afecta.



```
#include <stdio.h>
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
    printf("materia de computacion para ingenieros");
    system("PAUSE");
    return 0;
}
```

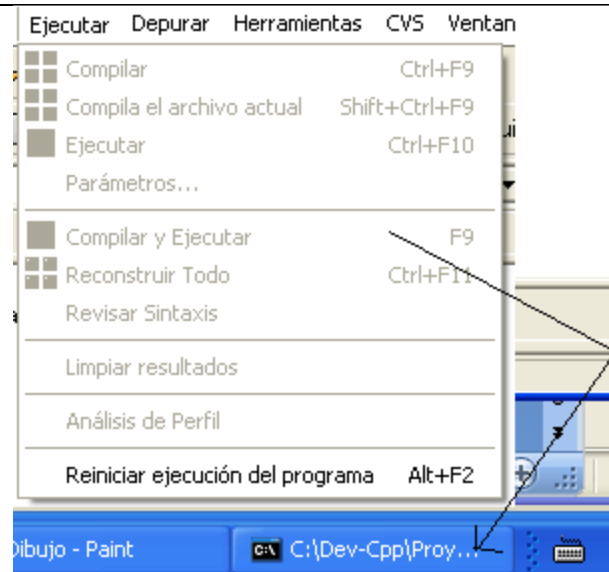
2. Después de cualquier modificación a nuestro programa DEBEMOS SIEMPRE **guardar los cambios**, apretando el icono de disquetes.



3. Compilar y correr con una solo acción. Solo con F9 o por menú.



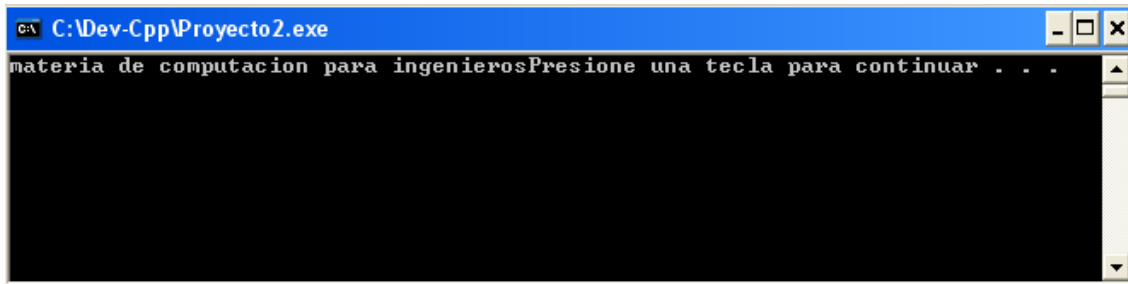
NOTA: SINO APARECIERA LA OPCION DE COMPILAR EJECUTAR EN ACTIVO :



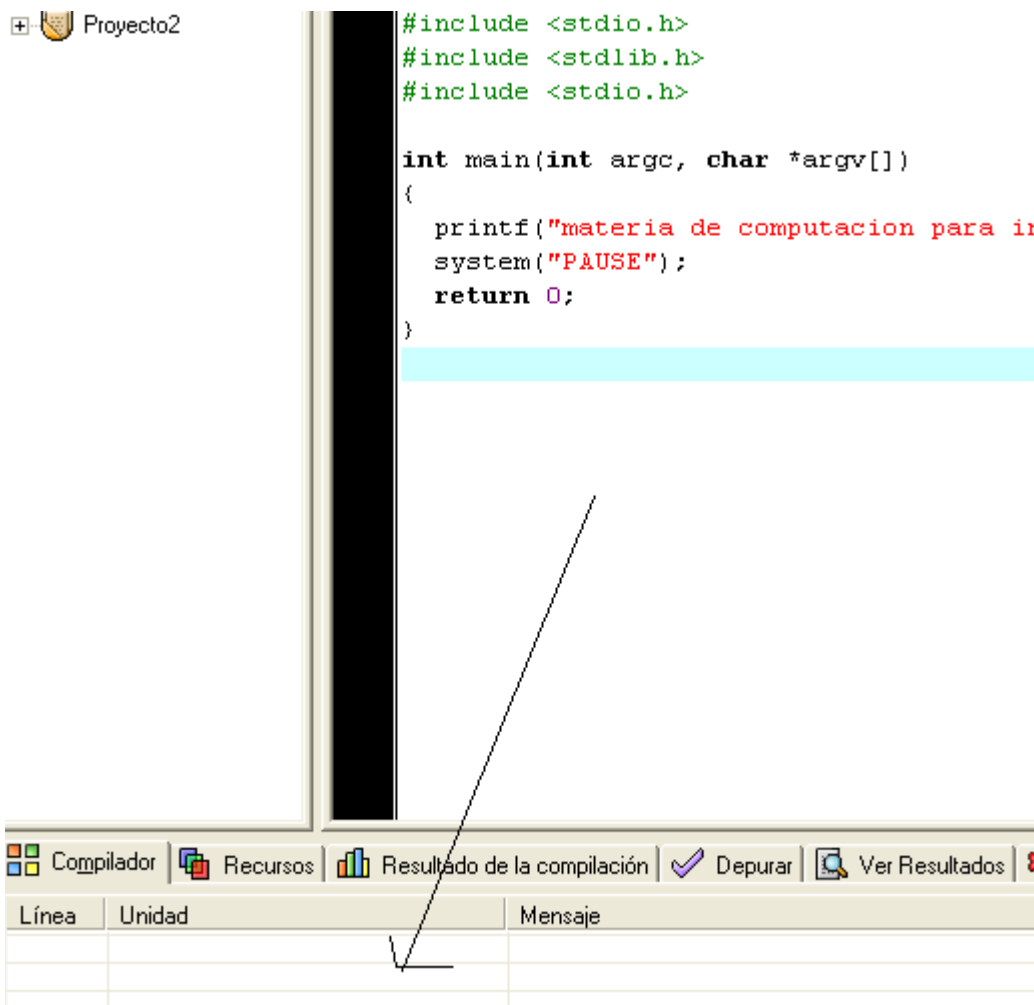
Significa que la corrida anterior NO LA CERRASTE, cualquier corrida de programa después de terminar de ver los resultados cierra la ventana con la cruz



Ahora veamos el resultado de la ejecucion:



Se ejecuto exitosamente y SIN Errores. Si hubiese errores aparecería en la pestaña del compilador con indicaciones, debajo de nuestro programa, en nuestro caso esta vacio; en el otro ejemplo generaremos un error, para que se note mas.



Parte 3

Programa con ERROR a propósito

Quita de la orden donde está el comando printf al final el punto y coma. Corre o ejecuta el programa, y checa que te avisa el compilador si existe error de sintaxis.

1. Quito punto y coma.

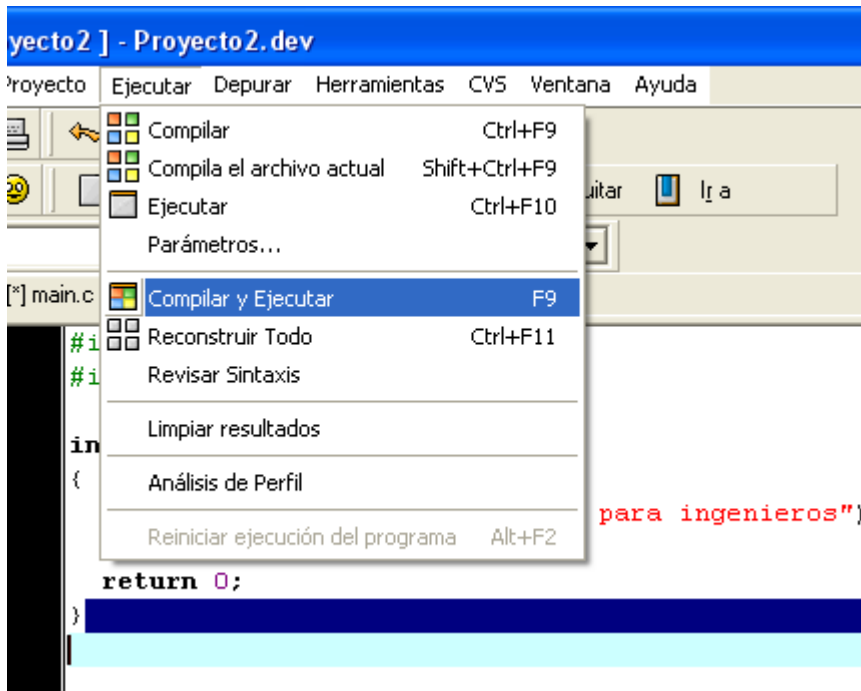
```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    printf("materia de computacion para ingenieros")
    system("PAUSE");
    return 0;
}
```

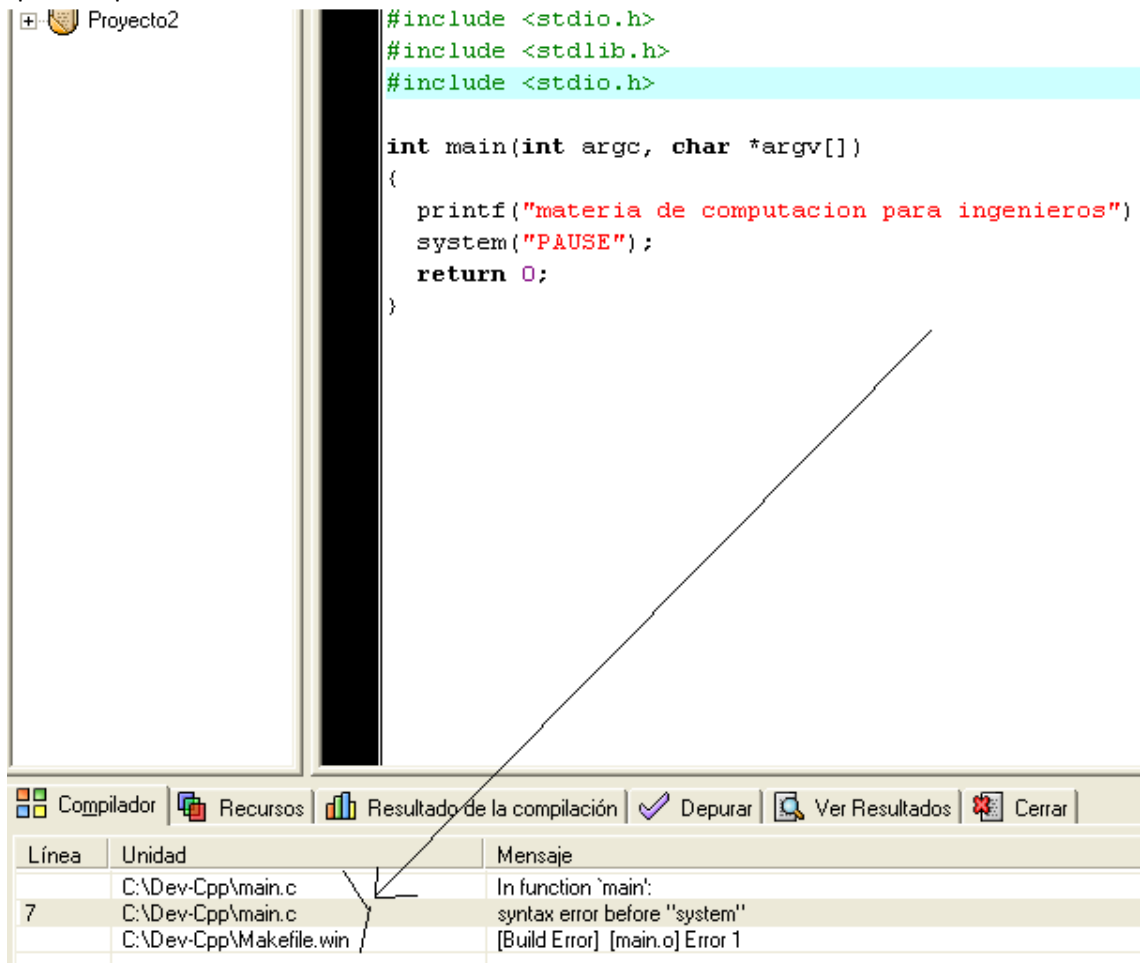
2. Después de cualquier modificación a nuestro programa DEBEMOS SIEMPRE **guardar los cambios**, apretando el icono de disquetes.



3. Compilar y correr con una solo acción. Solo con F9 o por menú.



- Para nuestro caso que omitimos el punto y coma, después de compilar y correr nos aparece que detecto el error.



- Corrigiendo nuestro error.**
Dice que Existe un error (tercera línea);

que está en la función main (primera línea).

Y la segunda línea dice: Before= antes; antes de system.

Tenemos que el anterior a "system" es el "printf"; debemos checar (SIEMPRE para cualquier error):

- a) ¿El comando esta en minúsculas o como se pide?
- b) ¿Tenemos un PAR de paréntesis de nuestro letrero, uno que abre otro que cierra? (Si hubiese muchos contenidos en una línea de orden hay que checar que hay el mismo número de abiertos que de cerrados)
- c) ¿Al final de cada orden tenemos un punto y coma (;)?

Ya sabemos que es el punto y coma, añadirlo por favor, guardar, compilar y correr. Ya debe de correr, pues ya no tiene errores.

FIN

Parte 4. Entendiendo la estructura del programa.

A. Cambiando SYSTEM

system(pause) nos permite detener nuestra pantalla de salida un determinado tiempo, hasta oprimir una tecla enviando un letrero que dice "Presiona una tecla para continuar...".

El comando system(pause): este produce el resultado

```
Presione una tecla para continuar . . . _
```

Si omitimos este comando no aparecerá el letrero, ni tampoco espera a que alguien apriete una tecla, y ejecutara las órdenes pedidas a una velocidad imperceptible por el ojo, cuando la computadora que se tenga sea veloz con las órdenes simples, como solo operaciones y letreros.

Opción 1. Si quisiéramos hacer esperar, hasta apretar una tecla SIN mostrar el letrero de system. Podemos utilizar: getch(); junto con la cabecera <conio.h> , donde se encuentra su función.

Observe como quedaría, adicione a su programa, guarde, compile y corra.


```
#include <stdio.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
```

```
int main(int argc, char *argv[])
```

```
    printf("materia de computacion para ingenieros");
    getch();
    return 0;
```

```
#include <stdio.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
```

```
main()
```

```
{
    printf("materia de computacion para ingenieros");
    getch();
    return 0;
}
```

```
C:\ C:\Dev-Cpp\Proyecto2.exe
```

```
materia de computacion para ingenieros_
```

```
#include <stdio.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
```

```
void main()
```

```
{
    printf("materia de computacion para ingenieros");
    getch();
}
```

```
C:\ C:\Dev-Cpp\Proyecto2.exe
```

```
materia de computacion para ingenieros
```

NOTA: UN PROGRAMA EN LENGUAJE C SIEMPRE CONTENDRA UNA FUNCION MINIMA (LA PRINCIPAL O MAIN), PUES PUEDE CONTENER ADEMAS PROGRAMAS ADJUNTOS LLAMADOS **FUNCIONES SECUNDARIAS** (EL MAIN SE LLAMA FUNCION PRINCIPAL).

Ejercicio: cambia el nombre de la función main , compílalo y ejecútalo, ¿qué pasa?

```

#include <stdio.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

principal()
{
    printf("materia de computacion para ingenieros");
    getch();
    return 0;
}

```

Línea	Unidad	Mensaje
		[Linker error] undefined reference to `WinMain@16'
		ld returned 1 exit status
	C:\Dev-Cpp\Makefile.win	[Build Error] [Proyecto2.exe] Error 1

Existe un error, SIN DEFINIR LA REFERENCIA A WinMain : NO ENCONTRO LA FUNCION PRINCIPAL, NO PUEDE CORRER EL PROGRAMA.

C. COLOCANDO LAS CABECERAS NECESARIAS EN BORLANDC PERO NO DEVCC++

El comando getch se encuentra dentro de un archivo de cabecera llamado "conio" la extensión .h significa header (cabecera), y se mandan a buscar a esa ruta con #include. RECUERDESE QUE LA CABECERA DEPENDE DEL COMPILADOR; EN EL COMPILADOR DE BORLAND ES NECESARIA PONER ESTA CABECERA, EN DEVCC, NO PERO ALGUNOS COMANDO NO FUNCIONA AQUÍ Y EN BORLAND SI.

Si omitiéramos la cabecera diría que no conoce el comando.(EN BORLANDC). En DEVCC tenemos una ventaja que dependiendo la versión y la instalación, se tiene rutas de las cabeceras más comunes (no todas) y no es necesario decir dónde buscarlas.

Pero antes de hacer este ejercicio, quitemos los archivos de cabecera que no estamos utilizando.

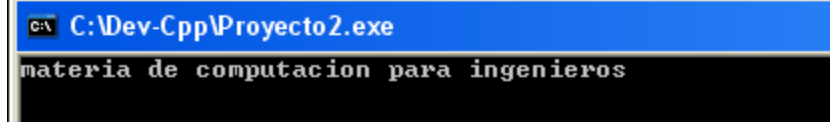
stdio.h significa cabecera estándar (st) de i(input= entrada) , o (output=salida), con ella funciona printf (significa imprimir archivo print file).

Stdlib.h=> cabecera librería estándar la cual trabajamos system (pero ya la quitamos).

Por lo tanto quitemos lo que nos sobra y corramos el programa.

```
#include <stdio.h>
#include <conio.h>

main()
{
    printf("materia de computacion para ingenieros");
    getch();
    return 0;
}
```



Estos dos #includes podrían omitirse en DEVC pero no en BorlandC.

RESUMEN

Ya sabemos cómo funcionan los comandos siguientes

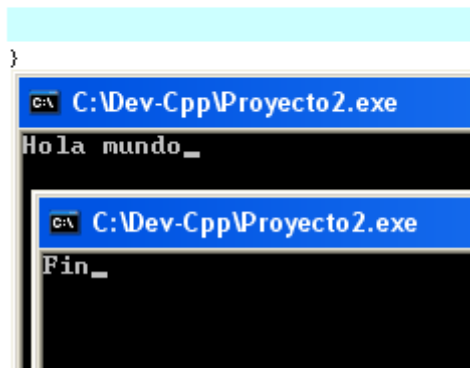
1. **system("PAUSE");** espera el oprimir tecla y manda mensaje en letrero.
2. **getch();** mantiene pantalla también, pero no lanza mensaje en letrero.
3. **Return 0;** se utiliza si la función es diferente de void, regresando un valor de 0.
4. **main() { }** Programa principal que siempre debe existir mínimo en un programa, no lleva punto y coma, y permite colocar nuestras ordenes dentro de paréntesis.
5. **printf("letrero");** aun cuando tiene muchos formatos que se verán después este formato básico permite enviar un mensaje de texto a la consola de salida (pantalla)
6. Cabeceras, se colocan para utilizar determinados comandos dentro del main o de funciones secundarias. Todas tienen extensión **.h**

PRACTICA 2

Parte 1. Comando. `system("cls");` Borra pantalla en BorlandC requiere `stdlib.h`.

Observe como se escribió y lo que hace. Escríbalo, Sálvelo y Ejecútelo. (ESE)

```
main()
{
    printf("Hola mundo");
    getch();
    system("cls");
    printf("Fin");
    getch();
}
```



Primero, imprime pantalla "hola mundo" con `printf`

Segundo, mantiene misma pantalla hasta que el usuario apriete alguna tecla, si lo hace entonces con `getch();` (Aprieta enter y observa)

Tercero, borra pantalla con `system("cls");`

Cuarto, imprime pantalla que dice FIN con `printf`

Quinto sostiene pantalla hasta oprimir tecla con `getch();`

Y termina

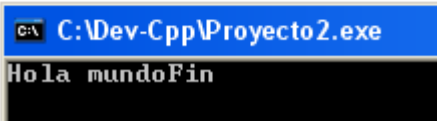
CABE RESALTAR QUE AQUÍ SE VEN DOS CONSOLAS DE SALIDAS SOLO ES UNA ES LA MISMA, SOLO QUE UNA ES ANTES DE BORRAR Y LA OTRA DESPUES DE BORRAR.

Parte 2. Modificando nuestro programa. Borremos antes de imprimir pantallas. (ESE)

```
main()
{
    system("cls");
    printf("Hola mundo");
    getch();

printf("Fin");
    getch();
}

```



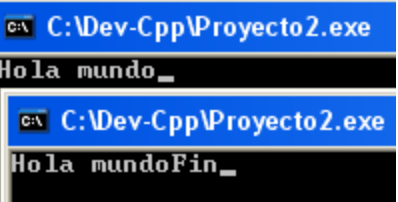
Para este caso, borro pantalla y después imprimió los dos letreros, los cuales se tuvo que oprimir enter para avanzar.

Parte 3. Modificando nuestro programa. Borremos al final del programa. (ESE)

```
main()
{
    printf("Hola mundo");
    getch();

printf("Fin");
    getch();
    system("cls");
}

```




Se ven los dos letreros juntos como el ejemplo anterior, y al final no se nota que la pantalla se borro porque al final de "fin" después de oprimir enter se termina el proceso.

Parte 4. Borremos antes de un getch

```
main()
{
    printf("Hola mundo");
    system("cls");
    getch();

    printf("Fin");
    getch();
}
```



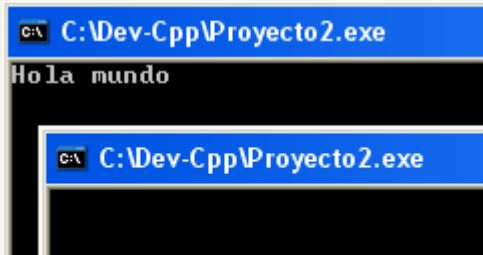
Como mandamos a borrar pantalla después de un letrero, pues ese letrero no lo vemos, solo alcanzamos a ver el último.

Parte 5. Borrar pantalla después del segundo letrero

```
main()
{
    printf("Hola mundo");

    getch();

    printf("Fin");
    system("cls");
    getch();
}
```



No se ve el segundo letrero.