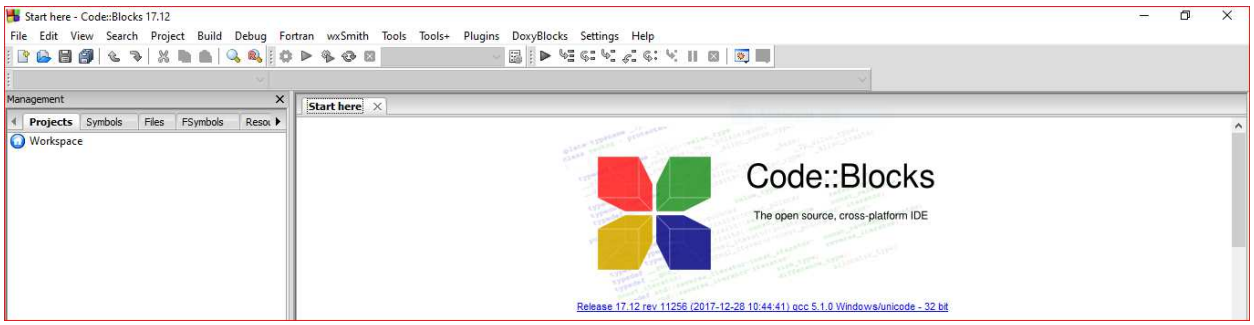
UNLaM
Dto. Ingeniería e Investigaciones Tecnológicas

Generar un proyecto para TDA

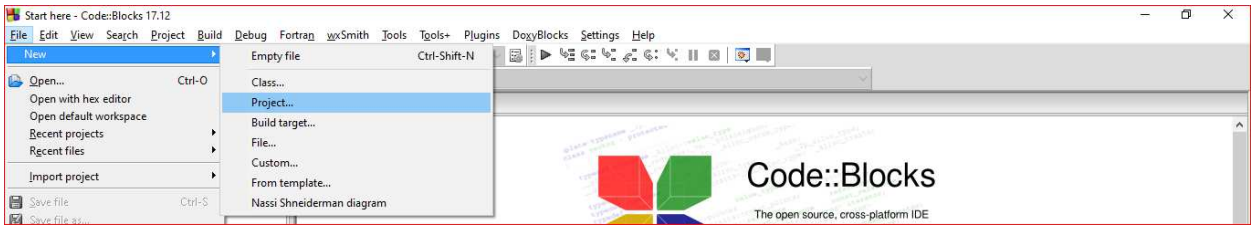
Dado que para trabajar con pilas y poder probar las primitivas de este TDA, vamos a necesitar cargar datos en la misma, comenzaremos por generar un proyecto e iremos generando nuestro entrono de prueba.

Comencemos abriendo el IDE de Code::Blocks . . .

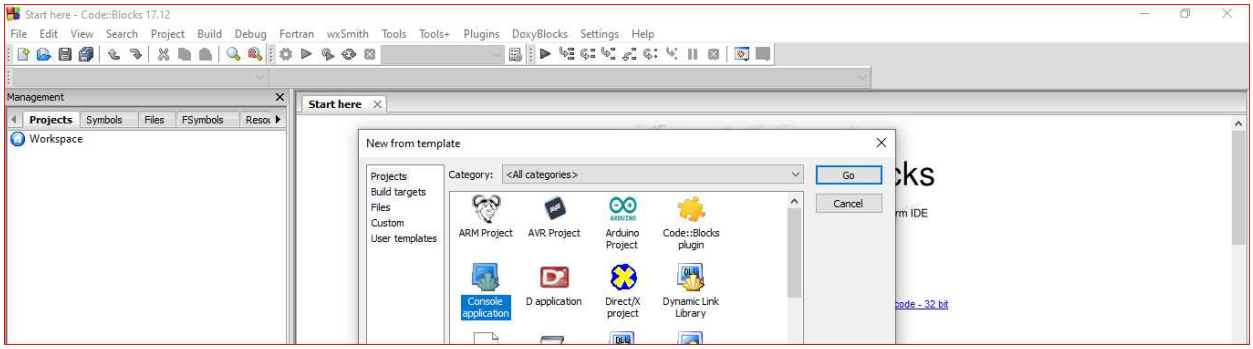


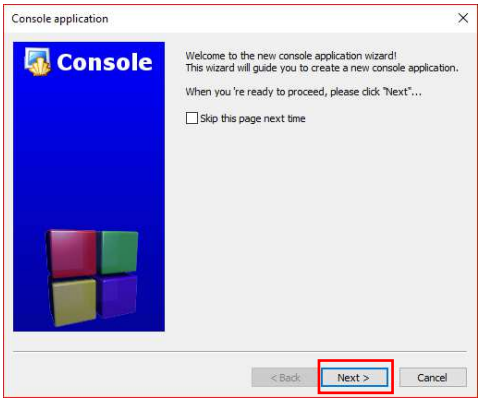
Elija dónde va a comenzar su proyecto, lo más recomendable es en una carpeta (o directorio), del disco **`C:\`** o en el escritorio de windows. Lo importante es que la ruta (o path), no sea demasiado largo, y que no tenga espacios en blanco. Comenzaremos por generar un proyecto con el nombre '**`pruebaPilaTDA_Din`**' en el subdirectorio 'TDA'.

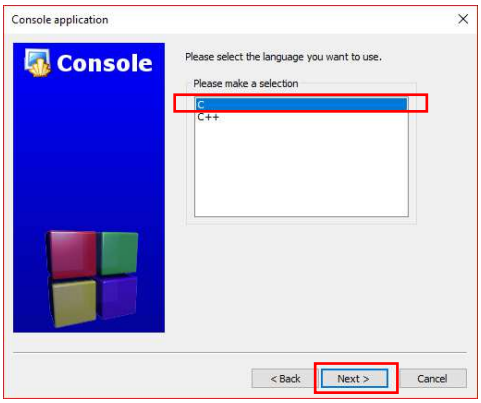
Entrando por el menú desplegable **`[File] / [New] / [Project …]`** (esto lo puede hacer sin el *mouse*, mediante la secuencia **`[Alt] + [F] – [N] – [P] – [Enter]`**).



. . . con lo que se abrirá la ventana emergente **`[New From Template]`**, en la que no modificaremos (en el panel izquierdo), lo seleccionado por el IDE (**`[Projects]`**). En el panel central seleccionaremos **`[Console application]`** y seguimos adelante mediante el botón **`[Go]`**.





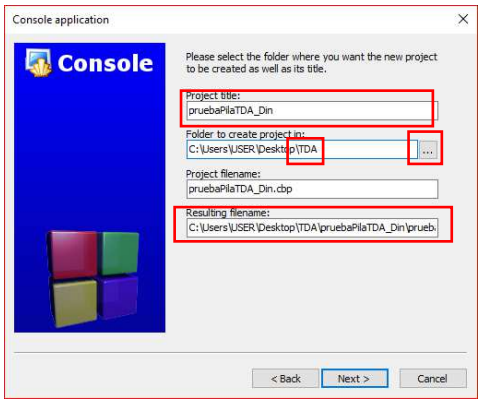


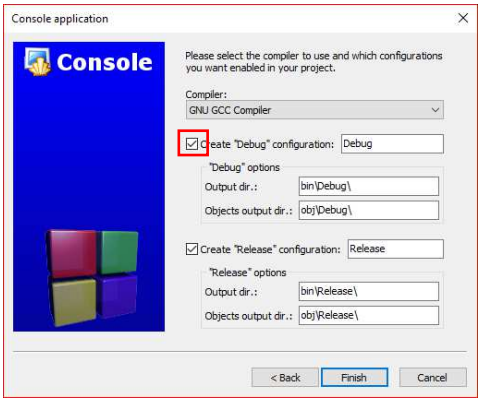
En la nueva ventana emergente (ver poco más abajo), mediante el botón **[…]** puede seleccionar una carpeta (directorio), para el proyecto o crearla. En este caso, tan solo seleccione, por ejemplo, el escritorio ("**C:\Users\USER\Desktop**"), para luego en el segundo diálogo de texto **[Folder to create Project in:]**, agregarle manualmente ("**\TDA**"), y en el primer diálogo **[Project title:]** indicarle el nombre del proyecto ("**pruebaPilaTDA_Din**").

Observe que en el cuarto diálogo **[Resulting filename:]** queda el *path name* completo del archivo del proyecto:

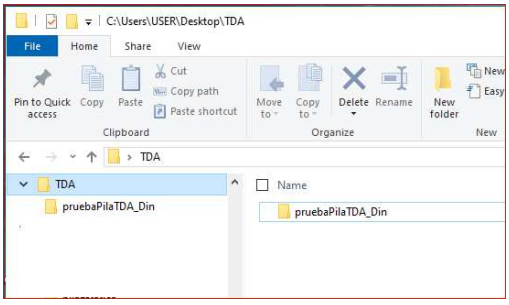
("**C:\Users\USER\Desktop\TDA\pruebaPilaTDA_Din\pruebaPilaTDA_Din.cbp** "), y en el tercero el nombre del proyecto (por ahora no modifique ni el tercer ni el cuarto diálogo de texto, ¡pruébelo en otro momento!). Finalmente siga adelante pulsando en

`[Next >]`. Se cambiará el contenido de la ventana emergente, en la que debe asegurarse, al menos, dejar seleccionado que se cree la configuración de depuración. Termine pulsando en el botón **`[Finish]`**.

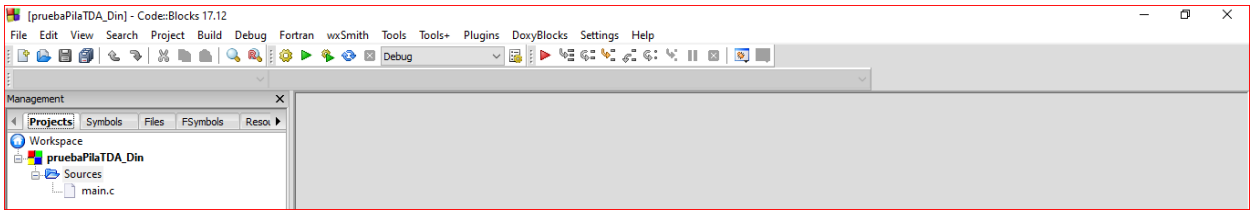




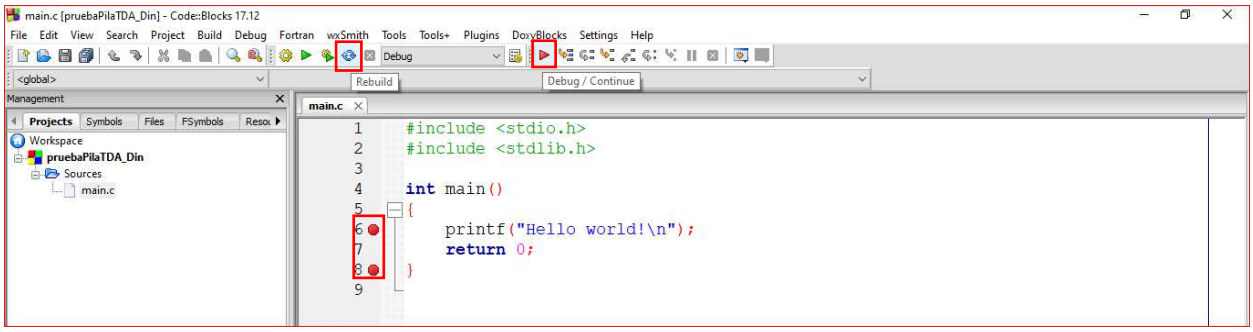
Minimice todas las ventanas y busque en el escritorio la *carpeta* creada por el entorno (yo la *arrastré* y la dejé ¡junto al *nadador*!). Hágale doble "click" a la *carpeta* e inspeccione el resultado que obtuvo. Mire también qué quedó en la carpeta "**`pruebaPilaTDA_Din`**".



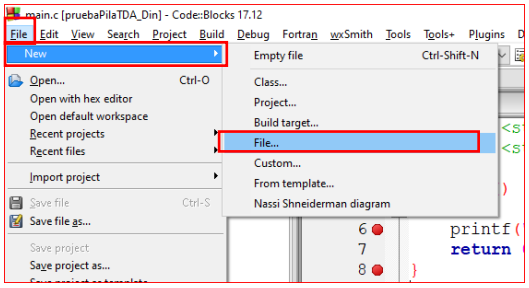
Maximice el IDE, verá el **`[Workspace]`** del que *cuelga* el proyecto, y del que a su vez *cuelgan* los "Sources" y a su vez, de este, *cuelga* el archivo "**`main.c`**". Hágale doble "click" a "**`main.c`**" con lo que verá el código fuente creado en forma mínima por el IDE.

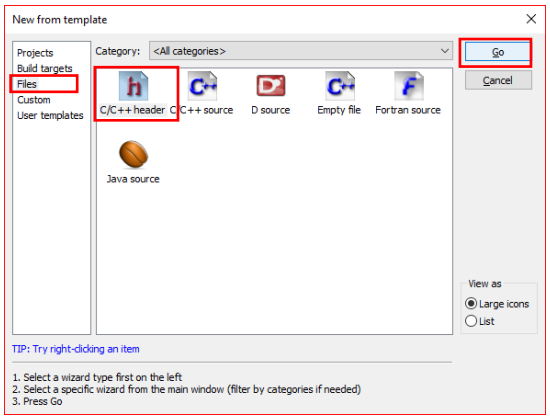


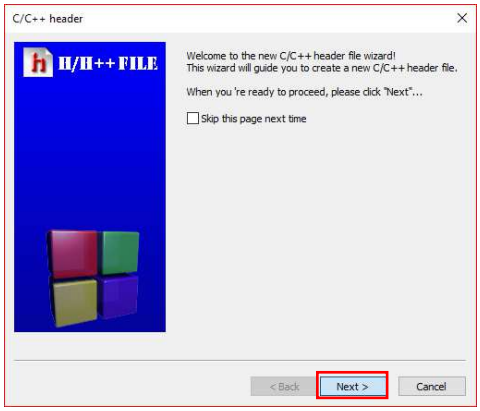
Asegúrese que el código compila (botón azul), ponga algunos *breakpoint*s haciendo "click" en el margen y proceda a ejecutar a modo *debug* (botón rojo). Vaya alternando entre la ventana de ejecución y el IDE.

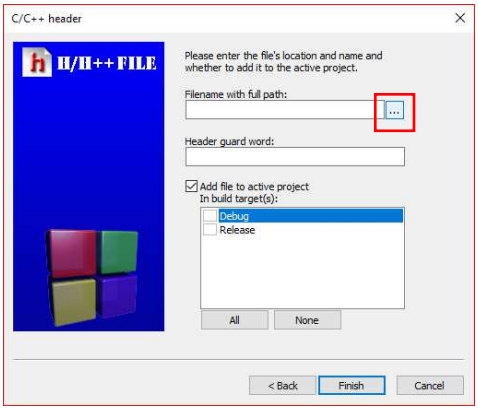


Proceda a agregar al proyecto el archivo "**`main.h`**", con los pasos indicados a continuación. (Recuerda la secuencia **`[Alt] + [F] – [N] – [F] – [Enter]`**).

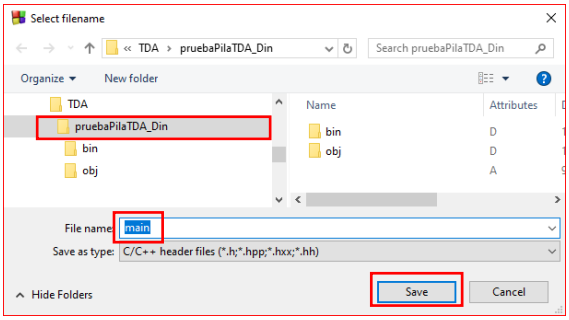


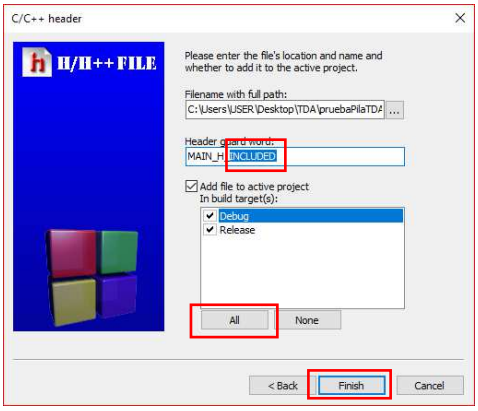




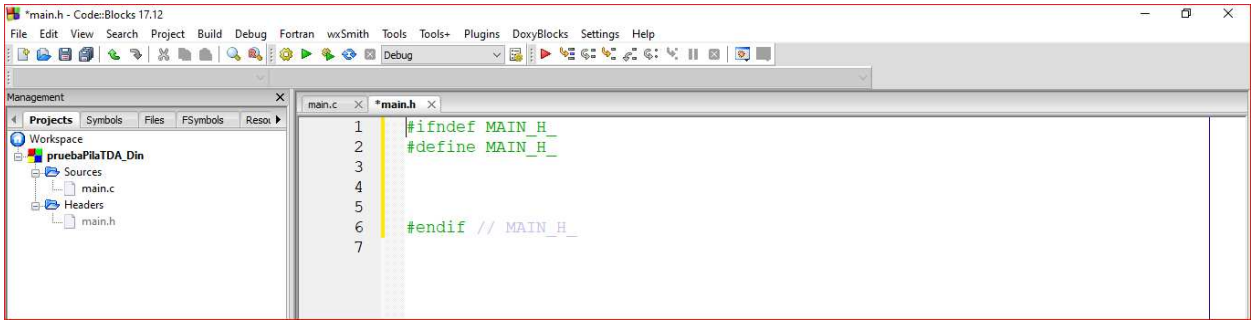


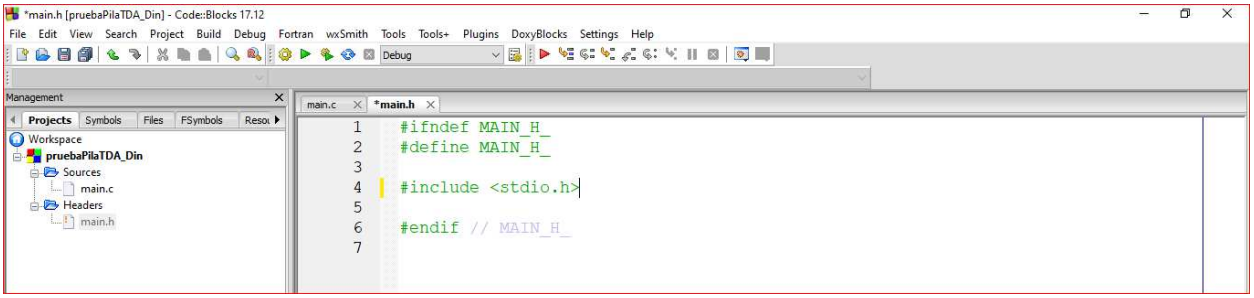
UNLaM
Dto. Ingeniería e Investigaciones Tecnológicas





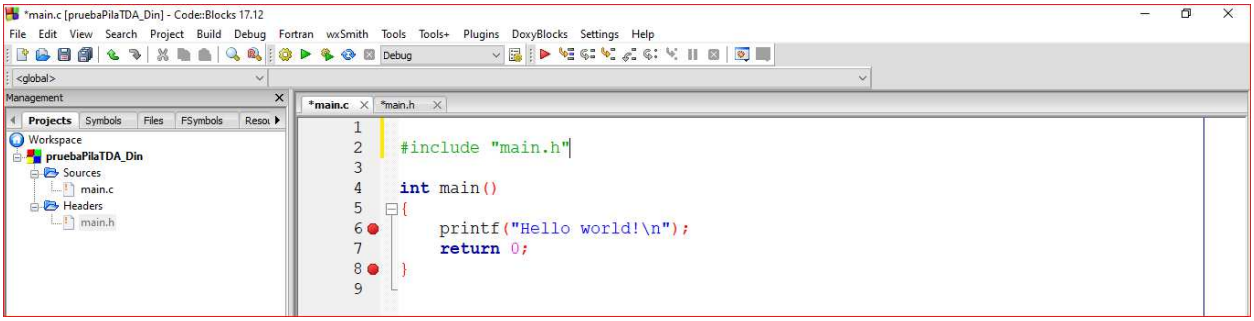
Tengo la *mala costumbre* de no dejarle que haga lo que quiera al IDE, y suelo eliminarle el "**`INCLUDED`**" a las etiquetas de compilación condicional . . . (ventana emergente **`[c/c++ header]`** de poco más arriba a la derecha). Una vez terminado, en el IDE quedará . . .

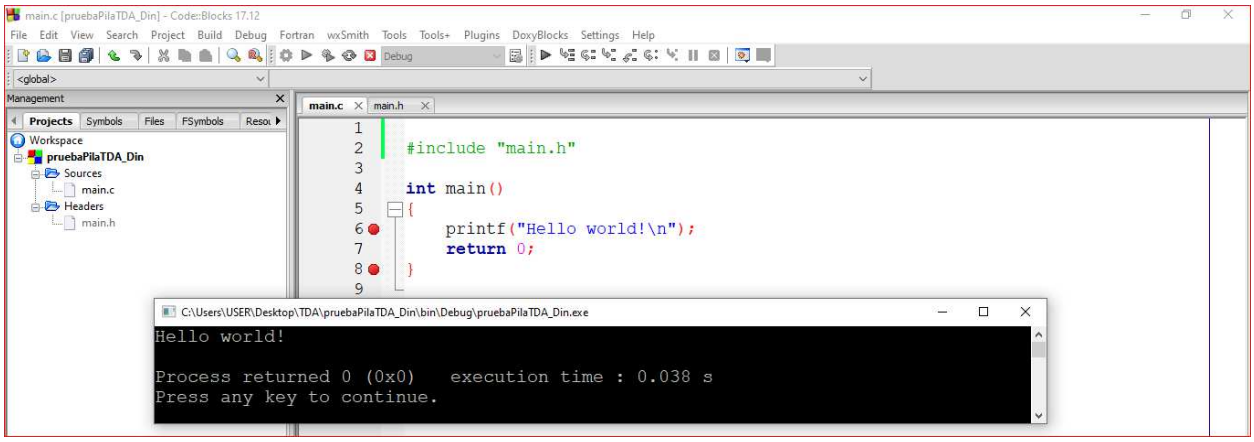




. . . y como en "**`main.c`**" se hace uso de la función "**`printf`**", le haremos el correspondiente *include* a "**`stdio.h`**" (ver arriba).

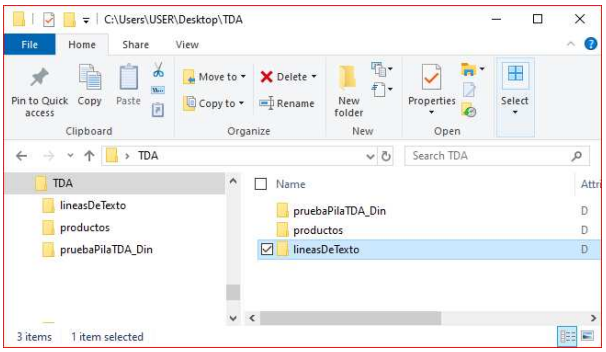
Pasando a la edición de "**`main.c`**", eliminaremos los *include* que hizo el IDE y sólo haremos *include* a "**`main.h`**". Compile y genere el ejecutable nuevamente (botón azul), asegurándose que *no hay errores ni advertencias* (*warnings*), y ejecute el proyecto.

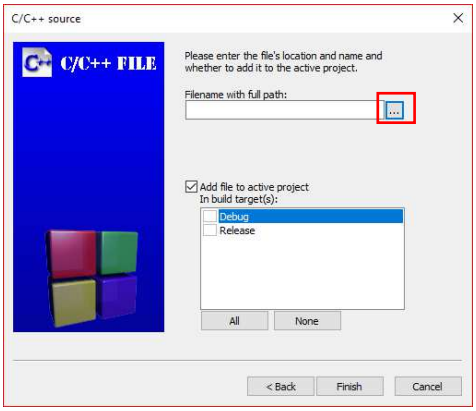


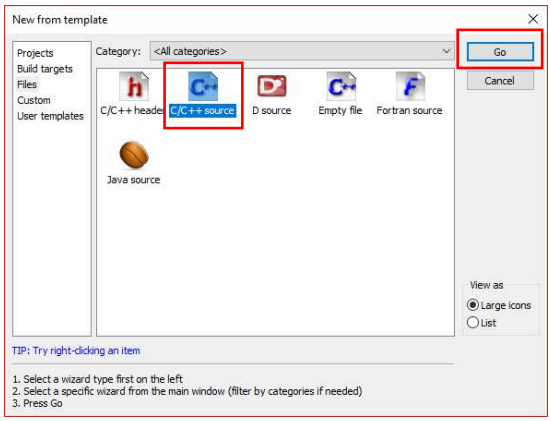


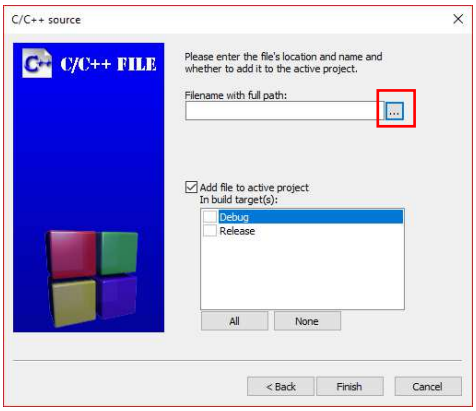
NOTA: Jamás termine la ejecución mediante el botón **`[X]`** en la barra de título a la derecha de la consola, tendrá problemas ya que dejará pendientes hilos de ejecución del programa. Puede hacerlo (con la ventana de ejecución activa pulsando cualquier tecla o de lo contrario (si estuviera en un bucle infinito), mediante **`[Ctrl] + [Pause/Break]`**.

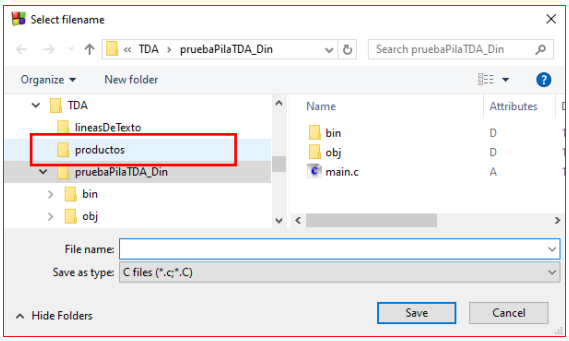
Pasemos a agregar en el proyecto los fuentes que nos permitirán ingresar información sintética (simulada). Por simplicidad, en la *carpeta* TDA, valiéndose del *explorador de windows*, agregue dos nuevas *carpetas* ("**`productos`**" y "**`lineasDeTexto`**"), tal como se ve en la próxima figura (izquierda). Luego agregue el fuente "**`productos.c`**" en la carpeta "**`productos`**" (ver la secuencia de figuras a continuación). ¿Recuerda la secuencia **`[Alt] + [F] – [N] – [F] – [Enter]`**?

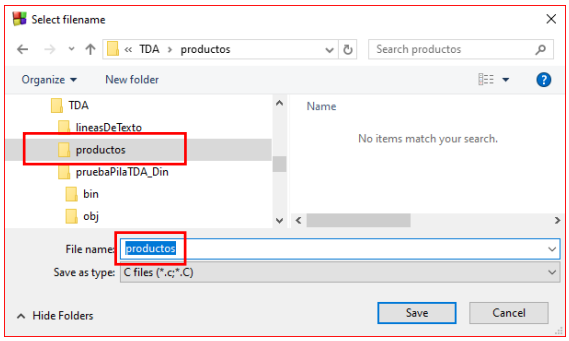

C:\Users\USER\Desktop\TDA
File
Home
Share
View
Pin to Quick access
Copy
Paste
Move to
Copy to
Delete
Rename
New folder
Properties
Select
Clipboard
Organize
New
Open
TDA
Search TDA
TDA
lineasDeTexto
productos
pruebaPilaTDA_Din
Name
pruebaPilaTDA_Din
productos
lineasDeTexto
3 items
1 item selected

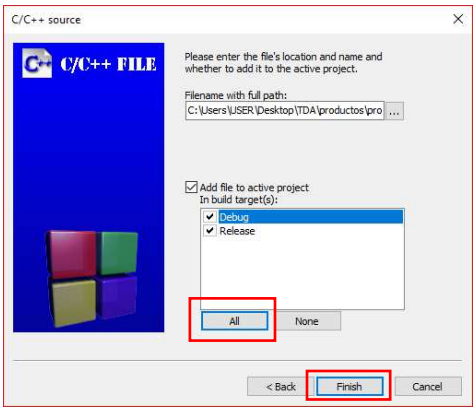

C/C++ source
C/C++ FILE
Please enter the file's location and name and whether to add it to the active project.
Filename with full path:
Add file to active project
In build target(s):
Debug
Release
All
None
< Back
Finish
Cancel

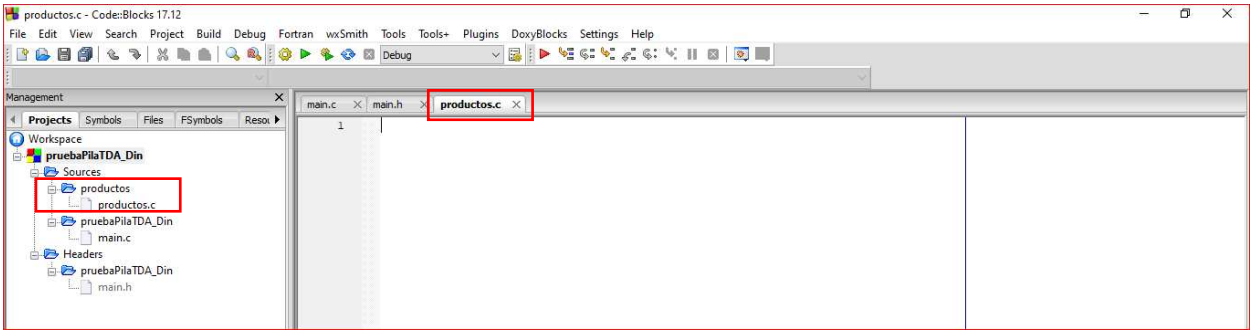

New from template
Projects
Build targets
Files
Custom
User templates
Category:
<All categories>
Go
Cancel
C/C++ header
C/C++ source
D source
Empty file
Fortran source
Java source
View as
Large icons
List
TIP: Try right-clicking an item
1. Select a wizard type first on the left
2. Select a specific wizard from the main window (filter by categories if needed)
3. Press Go


C/C++ source
C/C++ FILE
Please enter the file's location and name and whether to add it to the active project.
Filename with full path:
Add file to active project
In build target(s):
Debug
Release
All
None
< Back
Finish
Cancel

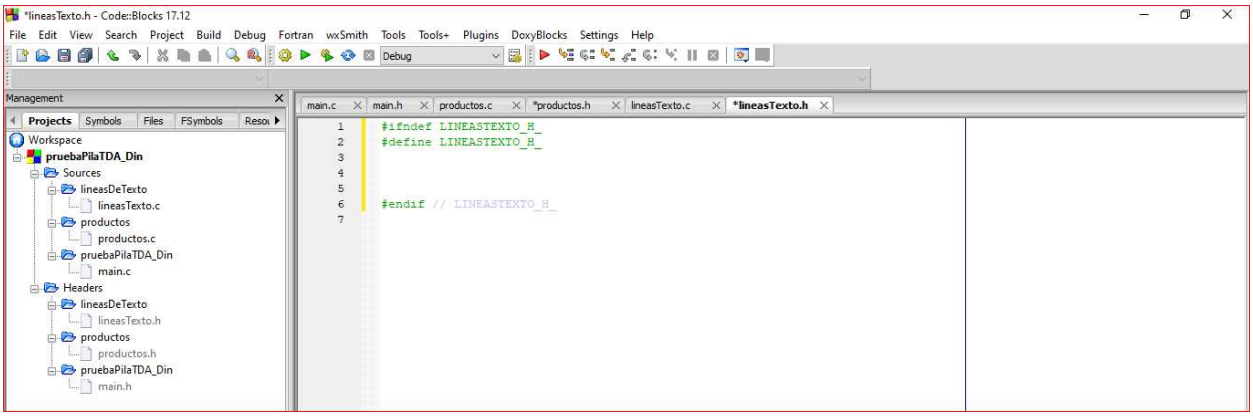

Select filename
TDA
pruebaPilaTDA_Din
Search pruebaPilaTDA_Din
Organize
New folder
TDA
lineasDeTexto
productos
pruebaPilaTDA_Din
bin
obj
Name
Attributes
bin
obj
main.c
File name:
Save as type:
C files (*.c;*.C)
Hide Folders
Save
Cancel


Select filename
TDA
productos
Search productos
Organize
New folder
TDA
lineasDeTexto
productos
pruebaPilaTDA_Din
bin
obj
Name
No items match your search.
File name:
productos
Save as type:
C files (*.c;*.C)
Hide Folders
Save
Cancel



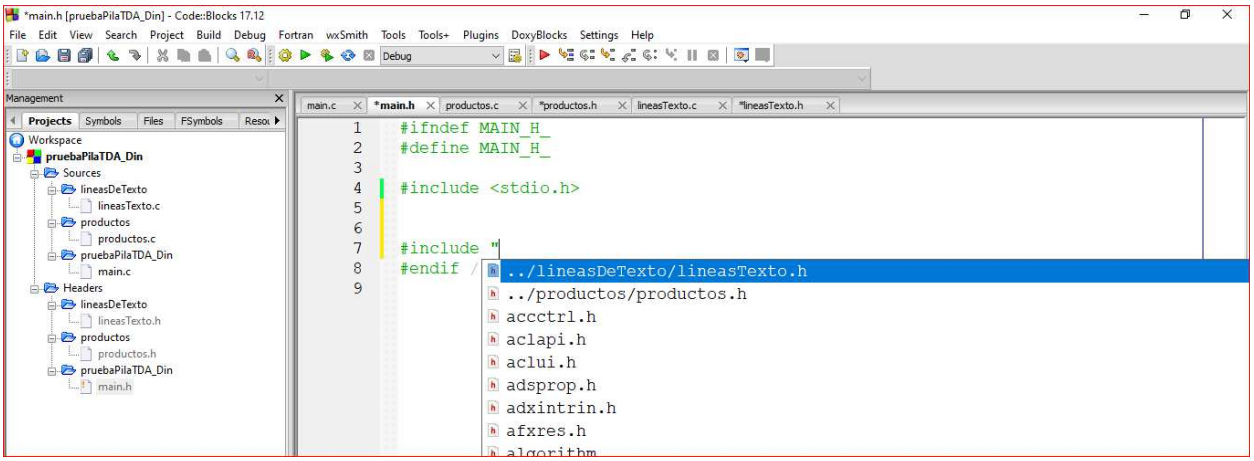


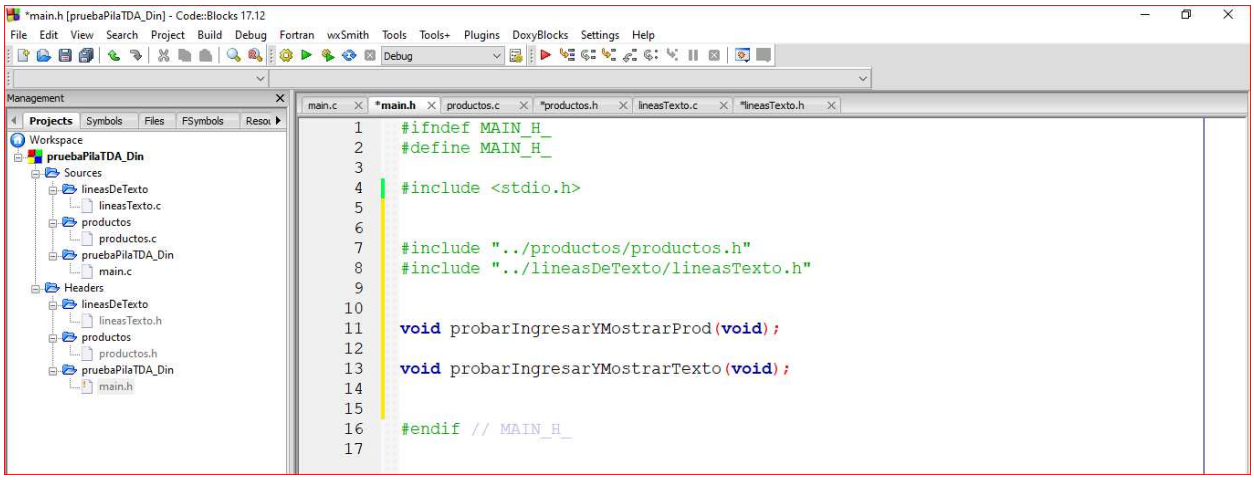
Siguiendo los mismos pasos con el IDE, agregue "**`productos.h`**" en la carpeta "**`productos`**"; y en la carpeta "**`lineasDeTexto`**" agregue los fuentes "**`lineaTexto.c`**" y "**`lineaTexto.h`**". Tras esto usted debería ver en el IDE …



Para luego poder utilizar en "**`main.c`**" los tipos de datos y funciones que luego codificaremos en los *módulos* recién creados, se procede a hacer los *include*

correspondientes en "**`main.h`**". Además, escriba los prototipos de "**`probarIngresarYMostrarProd`**" y de "**`probarIngresarYMostrarTexto`**", a desarrollar en "**`main.c`**" . . .

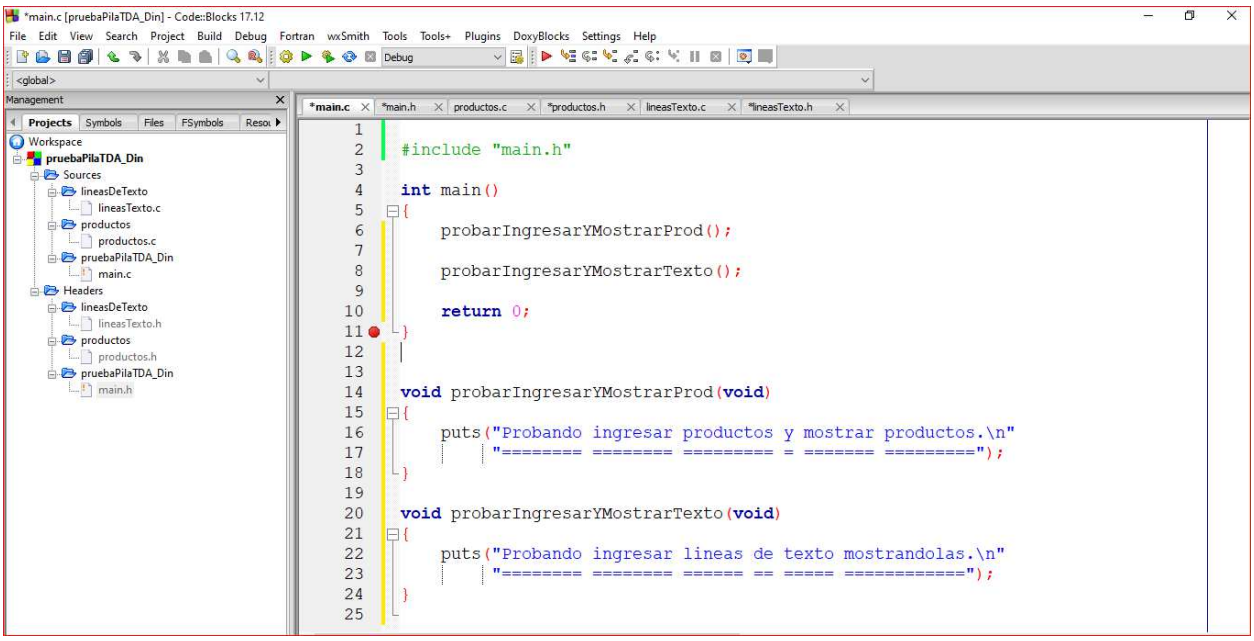


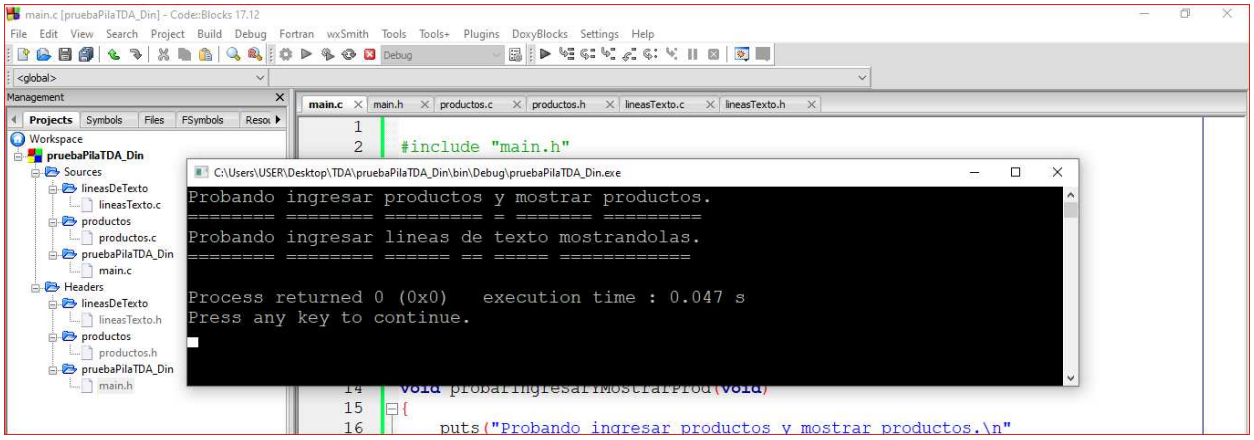


En la figura anterior se *ve* cómo queda "**`main.h`**".

Pase ahora a hacer un desarrollo mínimo de estas dos funciones en "**`main.c`**", para luego compilar y ejecutar (siempre es conveniente ir haciendo un desarrollo incremental), asegurándose de no haber introducido algo erróneo.

En la próxima figura se ve como queda "**`main.c`**" y en la siguiente el proyecto en ejecución.





Proceda a declarar y desarrollar el código correspondiente a "**`productos.h`**" y "**`productos.c`**", y luego a invocar desde "**`main.c`**" a las funciones (simuladas), de ingresar y mostrar productos.

UNLaM
Dto. Ingeniería e Investigaciones Tecnológicas

*productos.h - Code::Blocks 17.12

```
1  #ifndef PRODUCTOS_H_
2  #define PRODUCTOS_H_
3
4  #include <stdio.h>
5
6
7  typedef struct
8  {
9      char    codProd[11],
10             descrip[46];
11 } tProd;
12
13
14 int ingresarProducto(tProd *d);
15
16 void mostrarProducto(const tProd *d);
17
18
19 #endif // PRODUCTOS_H_
20
```

*productos.c - Code::Blocks 17.12

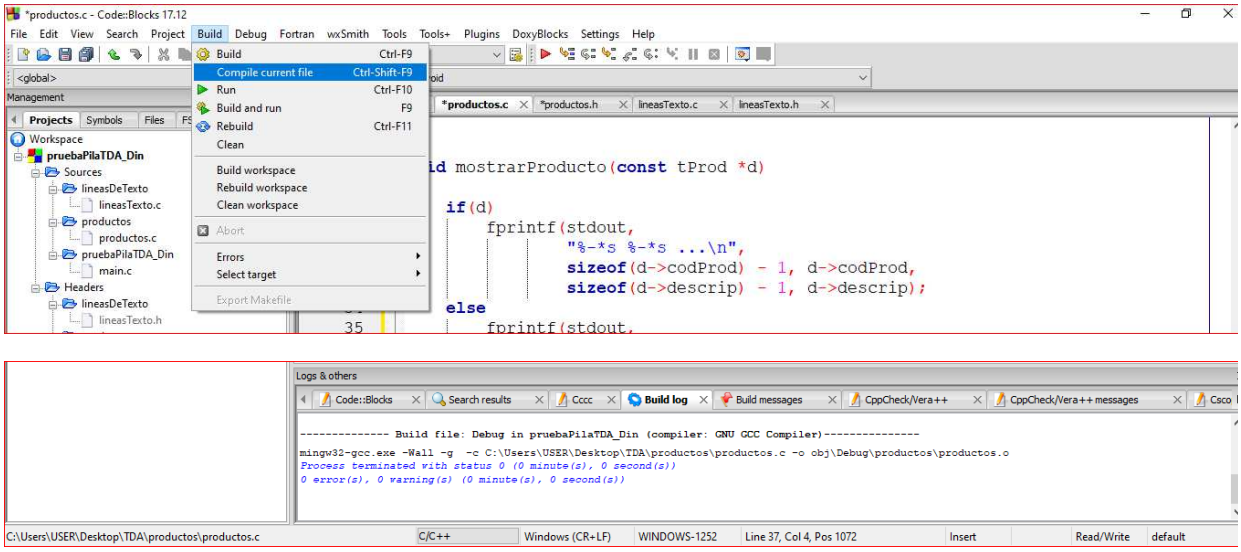
```
1
2  #include "productos.h"
3
4  int ingresarProducto(tProd *d)
5  {
6      static const tProd productos[] = {
7          ///1234567890    123456789 123456789 123456789 123456789 12345
8          { "clavoro3/4", "Clavo de oro 24 kilates de 3/4 de pulgada" },
9          { "martillo3K", "Martillo bolita con saca clavos de 3 kilos"},
10         { "alamyeso1",  "Alambre de yeso de un milimetro de espesor" },
11         { "rem-vid15",  "Remache de vidrio de 1,5 milimetros" },
12         { "plom-telgo", "Plomada de poliestireno expandido" },
13         { "limagoma17", "Lima de goma de 17 pulgadas"} };
14     static int posi = 0;
15
16     if(posi == sizeof(productos) / sizeof(tProd))
17     {
18         posi = 0;
19         return 0;
20     }
21     *d = productos[posi];
22     posi++;
23
24     return 1;
25 }
26
27 void mostrarProducto(const tProd *d)
28 {
29     if(d)
30         fprintf(stdout,
31                 "%-*s %-*s ...\n",
32                 sizeof(d->codProd) - 1, d->codProd,
33                 sizeof(d->descrip) - 1, d->descrip);
34     else
35         fprintf(stdout,
36                 "%-*.*s %-*.*s ...\n",
37                 sizeof(d->codProd) - 1, sizeof(d->codProd) - 1, "Cod. Producto",
38                 sizeof(d->descrip) - 1, sizeof(d->descrip) - 1,
39                     "Descripcion del producto");
40 }
41
```

NOTE que:

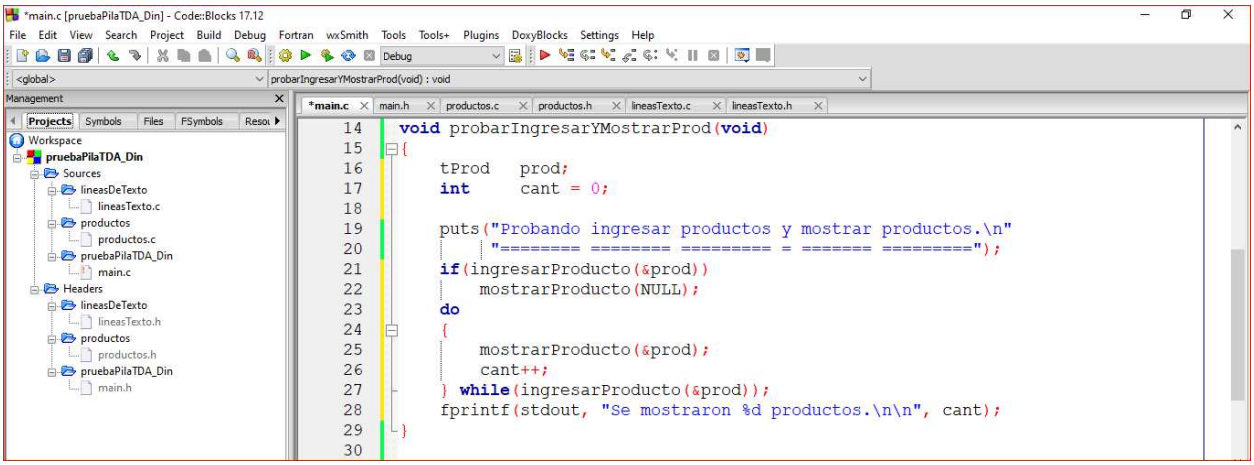
- la estrategia para mostrar el producto es que si recibe un puntero, muestra la información en un renglón, pero si recibe NULL, muestra los títulos en un renglón.

- la estrategia para la simulación del ingreso de información, es la de tener un array de estructuras pre inicializado con clave y descripción (por sencillez, sólo dos campos, pero podría tener más información); además de un entero que se utiliza de subíndice. Este entero se incrementa en uno cada vez que se invoca a la función y conserva ese valor para la próxima invocación. Cuando excede el máximo posible, se vuelve a poner en cero y devuelve un indicador de fracaso (cero), de lo contrario, devuelve un indicador de éxito (uno) y mediante su argumento (de salida), devuelve la información "*ingresada*".

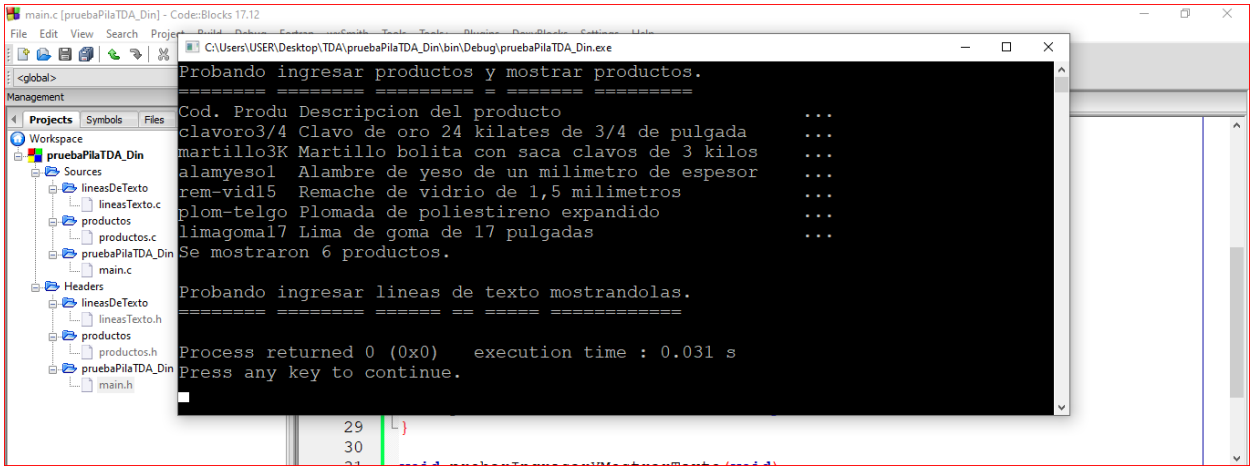
Si lo desea, puede proceder a compilar (sin generar el ejecutable), el fuente "**`productos.c`**" para asegurarse que todo está bien. Esto se hace con la ventana de "**`productos.c`**" activa en el IDE, mediante el menú desplegable **`[Build] / [Compile current file]`** (o mediante **`[Ctrl] + [Shift] + [F9]`**).



Finalmente probaremos el uso de estas dos funciones (las de ingresar y mostrar los productos). Edite "**`main.c`**" . . .



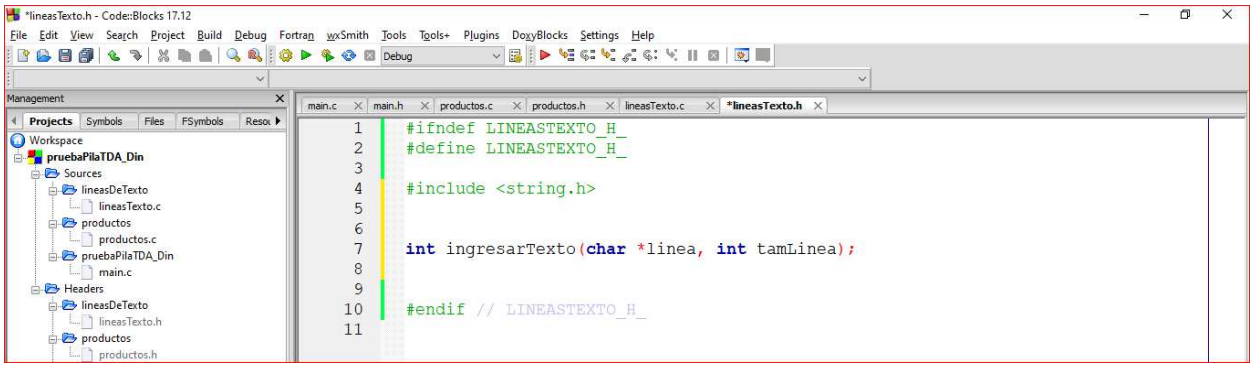
Compile y ejecute . . .

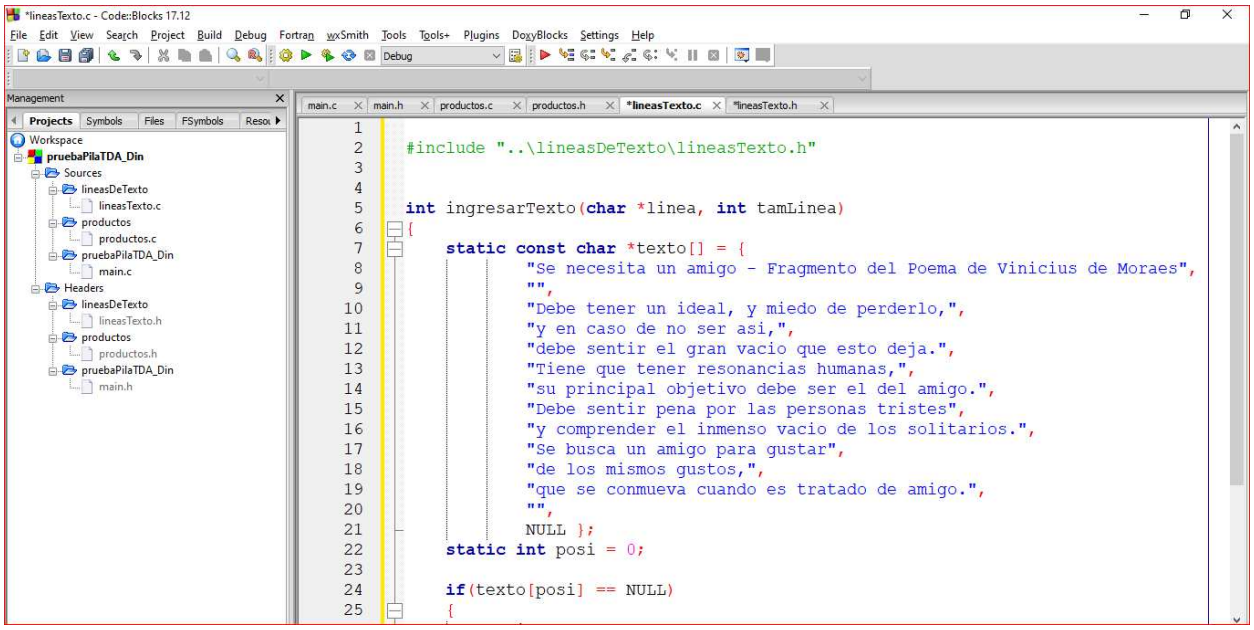


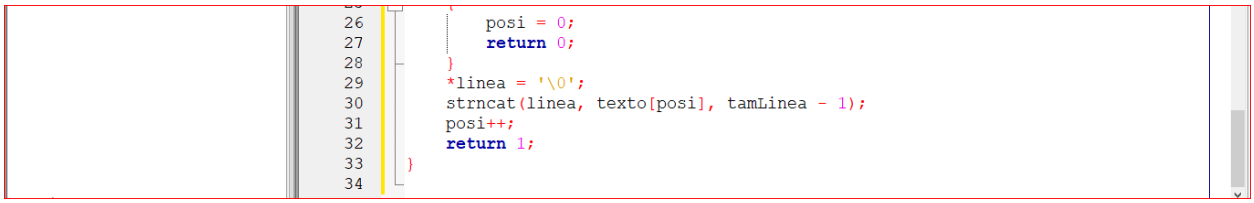
OBSERVACIÓN: El código de la función que acabamos de probar tiene un detalle (¡es un muy serio error!), que podría generar una violación de memoria. Diga cual es.

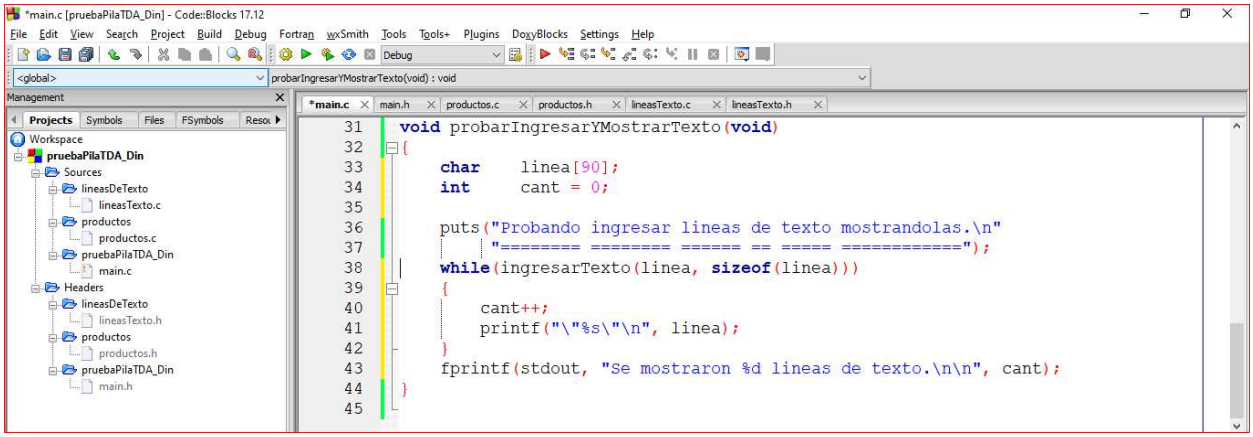
UNLaM
Dto. Ingeniería e Investigaciones Tecnológicas

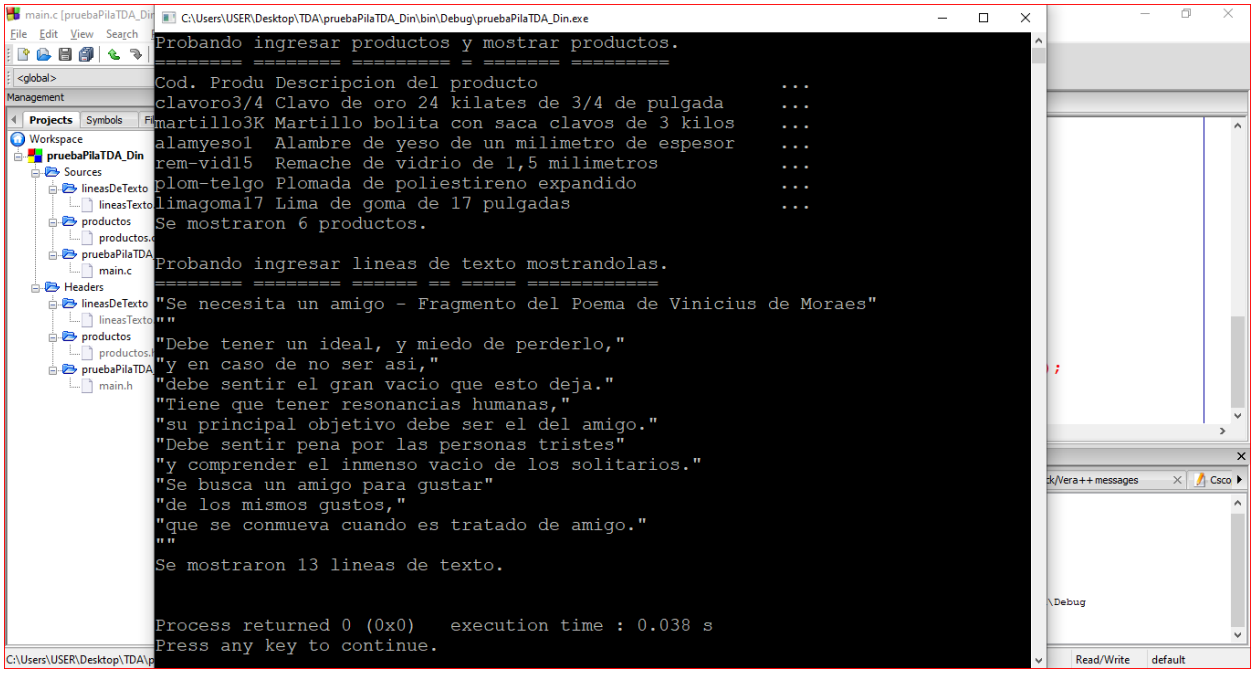
Complete el código de "**`ingresarTexto.h`**", "**`ingresarTexto.c`**" y de la prueba del ingreso de las líneas de texto en "**`main.c`**" y ejecute . . .











Continuaremos con este proyecto en cuanto avancemos con las primitivas del TDA PILA, y lo utilizaremos con los restantes TDA.