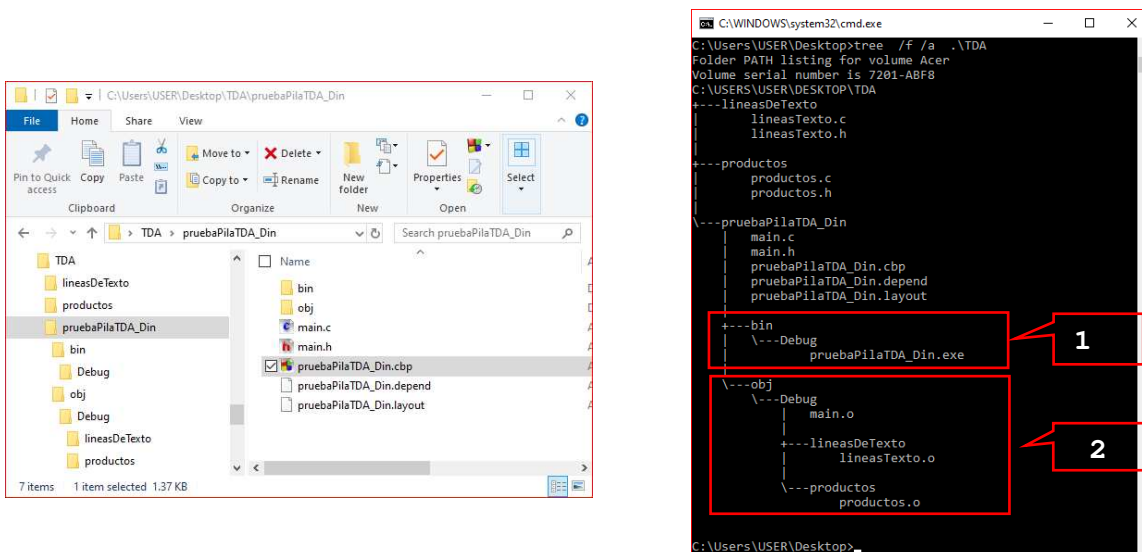


Continuar el proyecto para TDA

Ya se ha preparado el entorno de prueba y su entorno ha quedado (viéndolo con el explorador de Windows), como en la siguiente figura, tras expandir las *carpetas* :



La figura anterior a la derecha es el resultado de ejecutar el comando "**tree**" (son mañas de quién conoce algo de DOS y de haber trabajado alguna vez en Unix).

Note que el IDE al compilar, genera los archivos ".**depend**" y ".**layout**", además de las *carpetas* "**bin**" y "**obj**". En el primer archivo almacena información acerca de los distintos archivos fuente del proyecto además de otras características. En el segundo, almacena información de qué archivos están abiertos, cuál es el último que se estaba editando, en qué posición dentro de cada archivo estaba el cursor, etcétera. En el "**obj**", en *subcarpetas*, genera el resultado de compilar cada uno de los archivos fuente del proyecto. En "**bin**", en *subcarpetas*, los ejecutables para depuración [**Debug**] y para versión final [**Release**].

En cuanto al archivo ".**cbp**", cuando creamos el proyecto lo creó en IDE. Si lo abrimos con un editor de texto (p. ej.: **Notepad++** o si no hay más remedio el **Notepad** de



Windows), veremos que es un archivo en formato "xml" donde el IDE mantiene actualizados los archivos fuente que componen el proyecto . . .

```

1  <?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
2  <CodeBlocks_project_file>
3    <FileVersion major="1" minor="6" />
4    <Project>
5      <Option title="pruebaPilaTDA Din" />
6      <Option pch_mode="2" />
7      <Option compiler="gcc" />
8      <Build>
9        <Target title="Debug">
10          <Option output="bin/Debug/pruebaPilaTDA Din" prefix_auto="1" extension_auto="1" />
11          <Option object_output="obj/Debug/" />
12          <Option type="1" />
13          <Option compiler="gcc" />
14          <Compiler>
15            <Add option="-g" />
16          </Compiler>
17        </Target>
18        <Target title="Release">
19          <Option output="bin/Release/pruebaPilaTDA Din" prefix_auto="1" extension_auto="1" />
20          <Option object_output="obj/Release/" />
21          <Option type="1" />
22          <Option compiler="gcc" />
23          <Compiler>
24            <Add option="-O2" />
25          </Compiler>
26          <Linker>
27            <Add option="-s" />
28          </Linker>
29        </Target>
30      </Build>
31      <Compiler>
32        <Add option="-Wall" />
33      </Compiler>
34      <Unit filename=" ../lineasDeTexto/lineasTexto.c" >
35        <Option compilerVar="CC" />
36      </Unit>
37      <Unit filename=" ../lineasDeTexto/lineasTexto.h" >
38      </Unit>
39      <Unit filename=" ../productos/productos.c" >
40        <Option compilerVar="CC" />
41      </Unit>
42      <Unit filename=" ../productos/productos.h" >
43      </Unit>
44      <Unit filename="main.c" >
45        <Option compilerVar="CC" />
46      </Unit>
47      <Unit filename="main.h" />
48      </Unit>
49      <Extensions>
50        <code_completion />
51        <envvars />
52        <debugger />
53      </Extensions>
54    </Project>
55  </CodeBlocks_project_file>

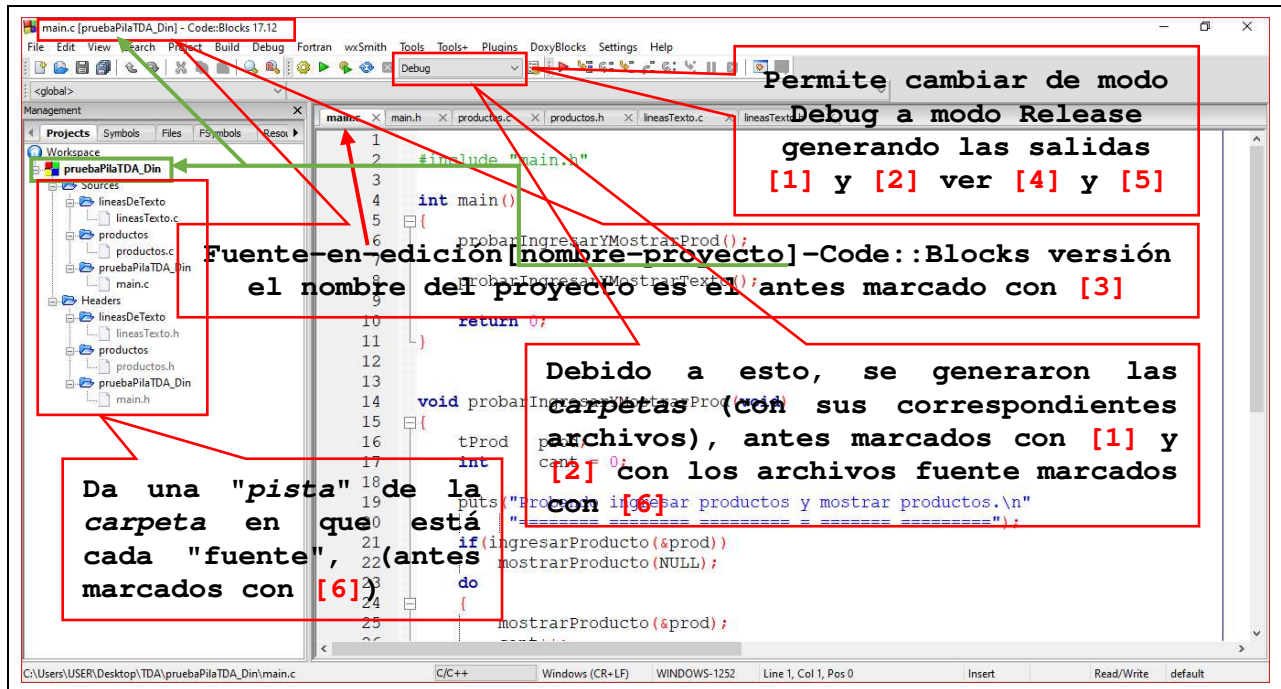
```

Cuando abrimos este archivo (haciéndole "doble click" con el mouse, lo abre el IDE de Code::Blocks), con lo que se abrirá el proyecto con los fuentes que hemos estado editando . . .



UNLaM

Dto. Ingeniería e Investigaciones Tecnológicas

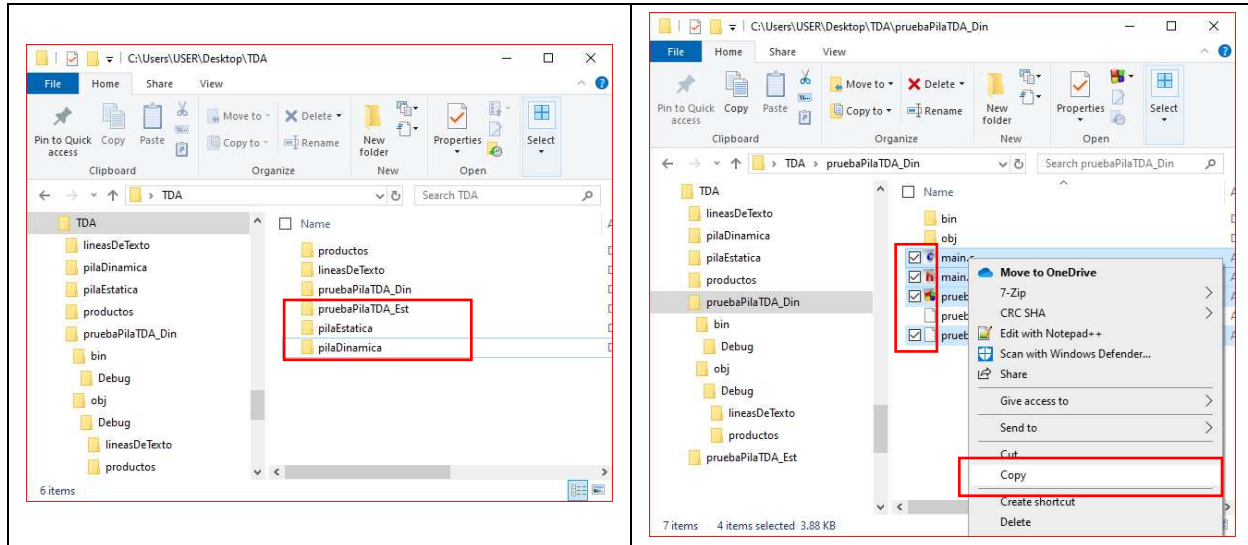


Continuemos con lo nuestro. El objetivo que tenemos es escribir el código de las implementaciones estática y dinámica del TDA Pila, y a usted le parece que le estoy dando una explicación de archivos con formato "xml" y de comandos del DOS. Yo le diría que no todo es "plug & play", que hay otro mundo, pero está muy bueno tener el "plug & play".

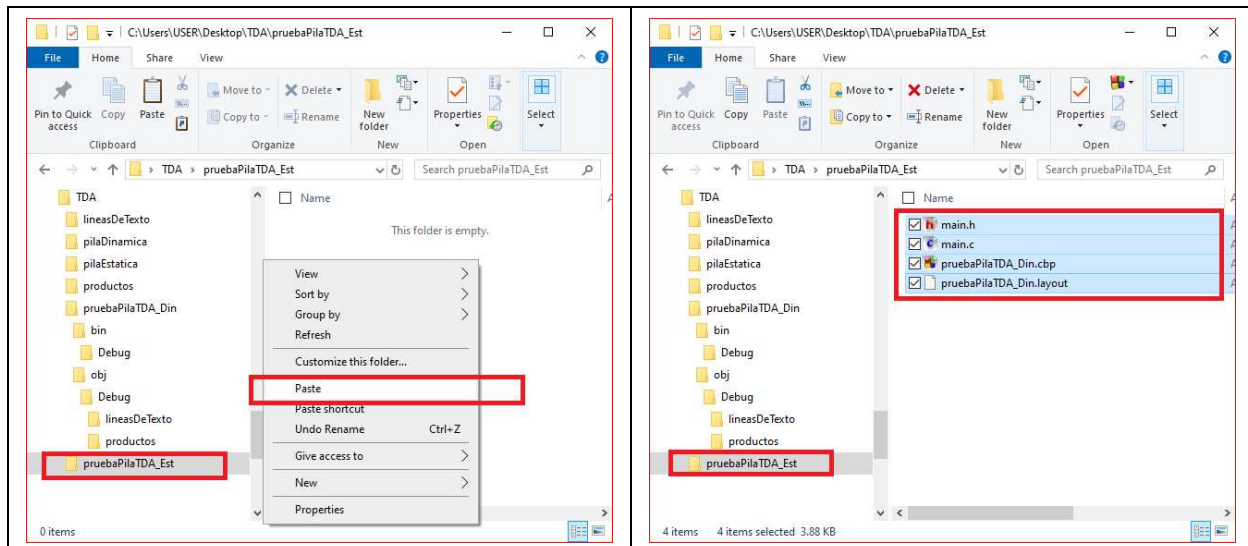
Lo que vamos a hacer para no complicarnos con el IDE, es generar una *carpeta* para el proyecto que utiliza la implementación con asignación estática de memoria y dejar este para la asignación dinámica. Además, tener los archivos fuente de pila estática y pila dinámica en *carpetas* separadas.

Así que valiéndonos del explorador de Windows vamos a generar una *carpeta* con el nombre "pruebaPilaTDA_Est", y otras dos carpetas con los nombres "pilaDinamica" y "pilaEstatica", todo dentro de la *carpeta* "TDA".

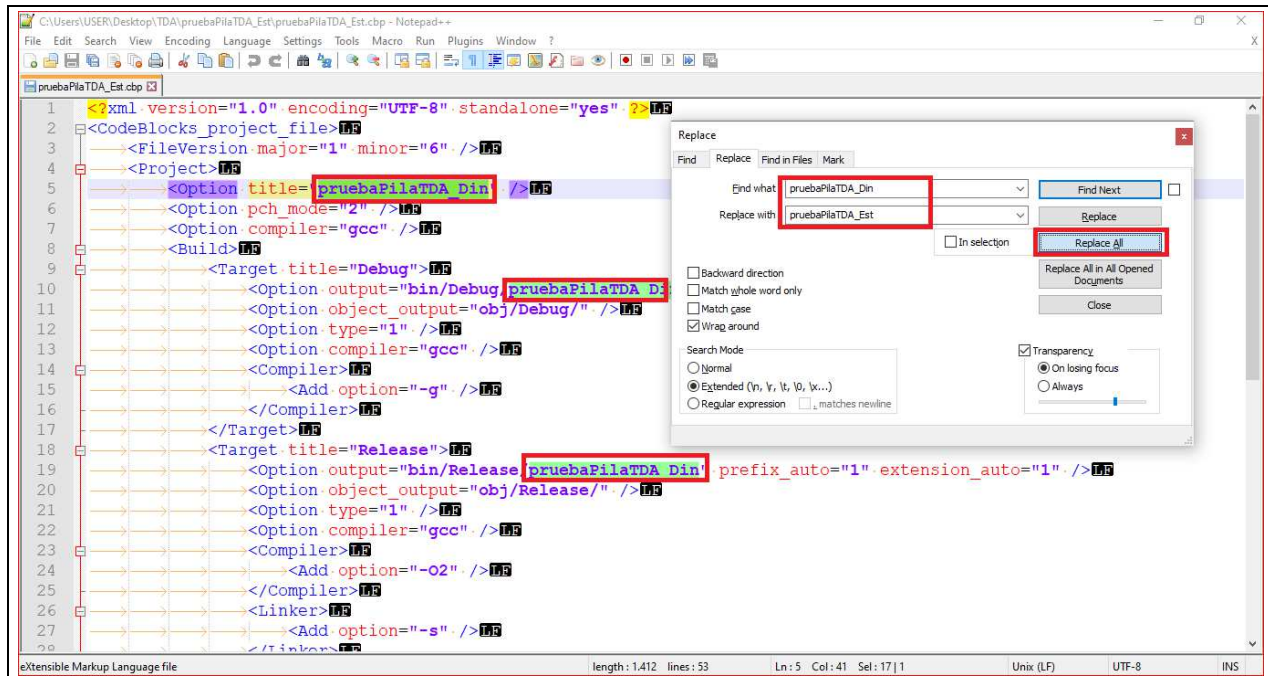
Vaya a la *carpeta* "pruebaPilaTDA_Din" y seleccione los cuatro archivos "main.c", "main.h", "pruebaPilaTDA_Din.cbp" y "pruebaPilaTDA_Din.layout" para ...



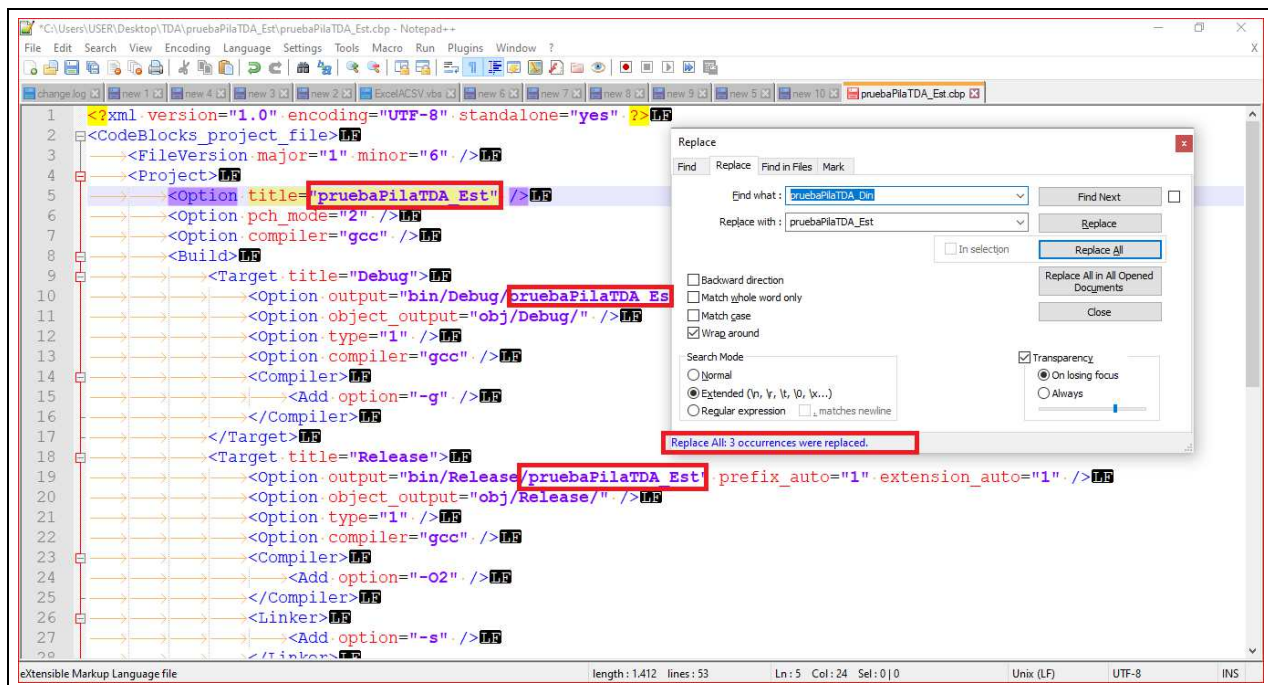
... luego ir a la carpeta "**pruebaPilaTDA_Est**", y pegarlos en ella. . .



Y ahora con muy poco esfuerzo, modificamos el archivo del proyecto y del diseño del proyecto (".cbp" y ".layout"). Seleccione el archivo recién "*pegado*" (cualquiera de los dos), y renómbrelo (modificando la terminación "**Din**" y reemplázela por "**Est**", y ya que está copie el nombre de archivo ("**pruebaPilaTDA_Est**"). Renombre el otro archivo pegando directamente el nombre (manteniendo la extensión). Luego abra el archivo del proyecto ("**pruebaPilaTDA_Est.cbp**"), con el Notepad (o mejor con Notepad++) . . .



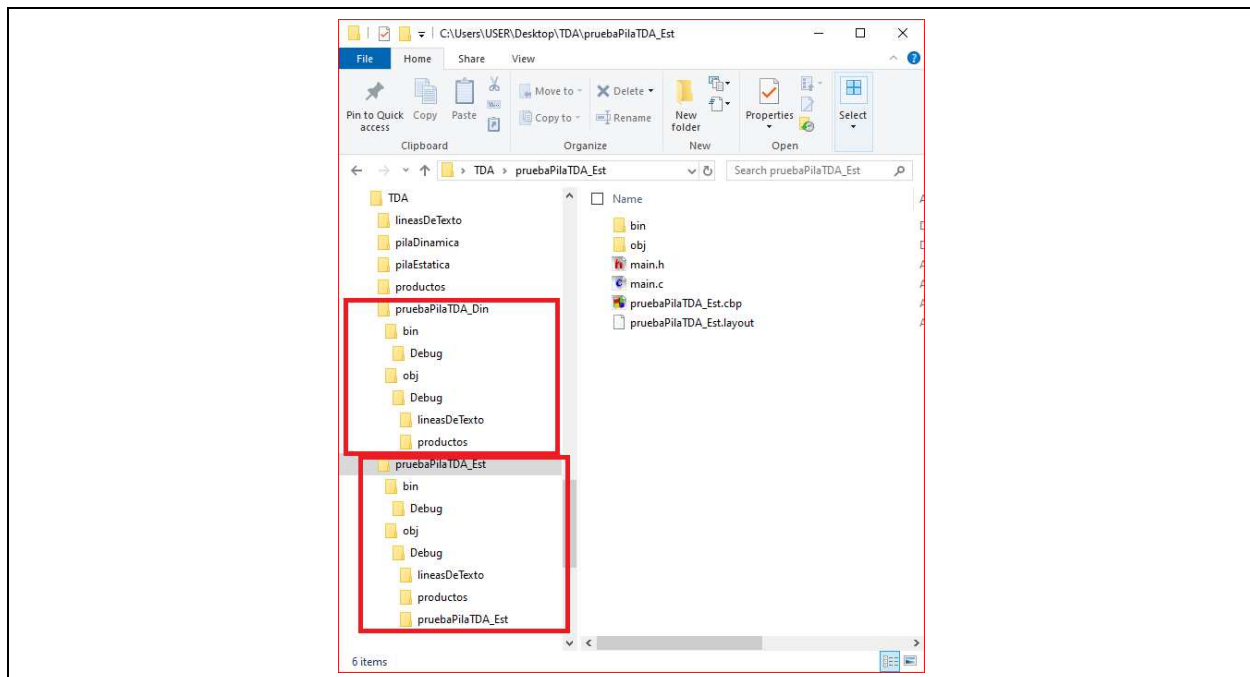
Lo único que vamos a hacer es reemplazar todas las ocurrencias (dos o tres), de "pruebaPilaTDA_Din.cbp" por "pruebaPilaTDA_Est.cbp".



Todo lo que hemos hecho es mostrarle cómo se ahorrará algún tiempo de trabajo con el IDE, que en este caso, dado que son proyectos repetidos (se usarán las primitivas de una o de otra Pila), con los mismos tipos de información (y además cuando avancemos con los otros TDA), podremos ahorrar tiempo.

Abra el proyecto que acabamos de generar (copiando y pegando), compílelo y ejecútelo, viendo que funciona del mismo modo que el original.

Compruebe que le generó un árbol de *carpetas* idéntico al del primer proyecto.

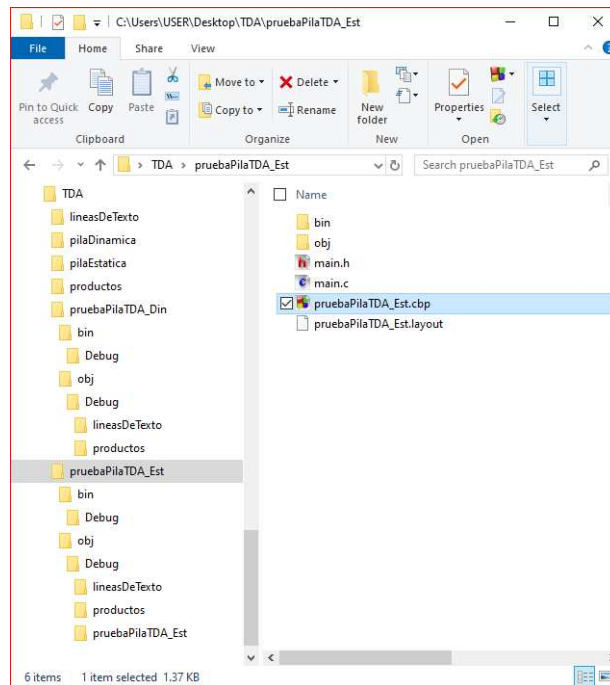


Todo lo que falta es crear las correspondientes implementaciones del TDA Pila.

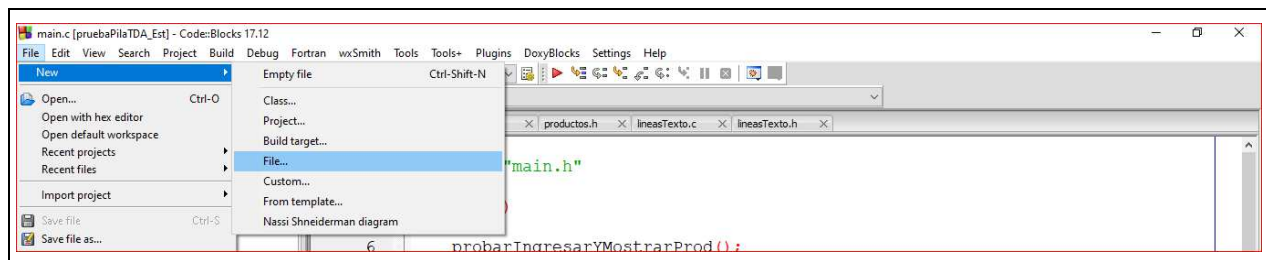
Comencemos por una de las dos, por ejemplo, y ya que lo tenemos recién *creado*, el proyecto de prueba de pila estática.

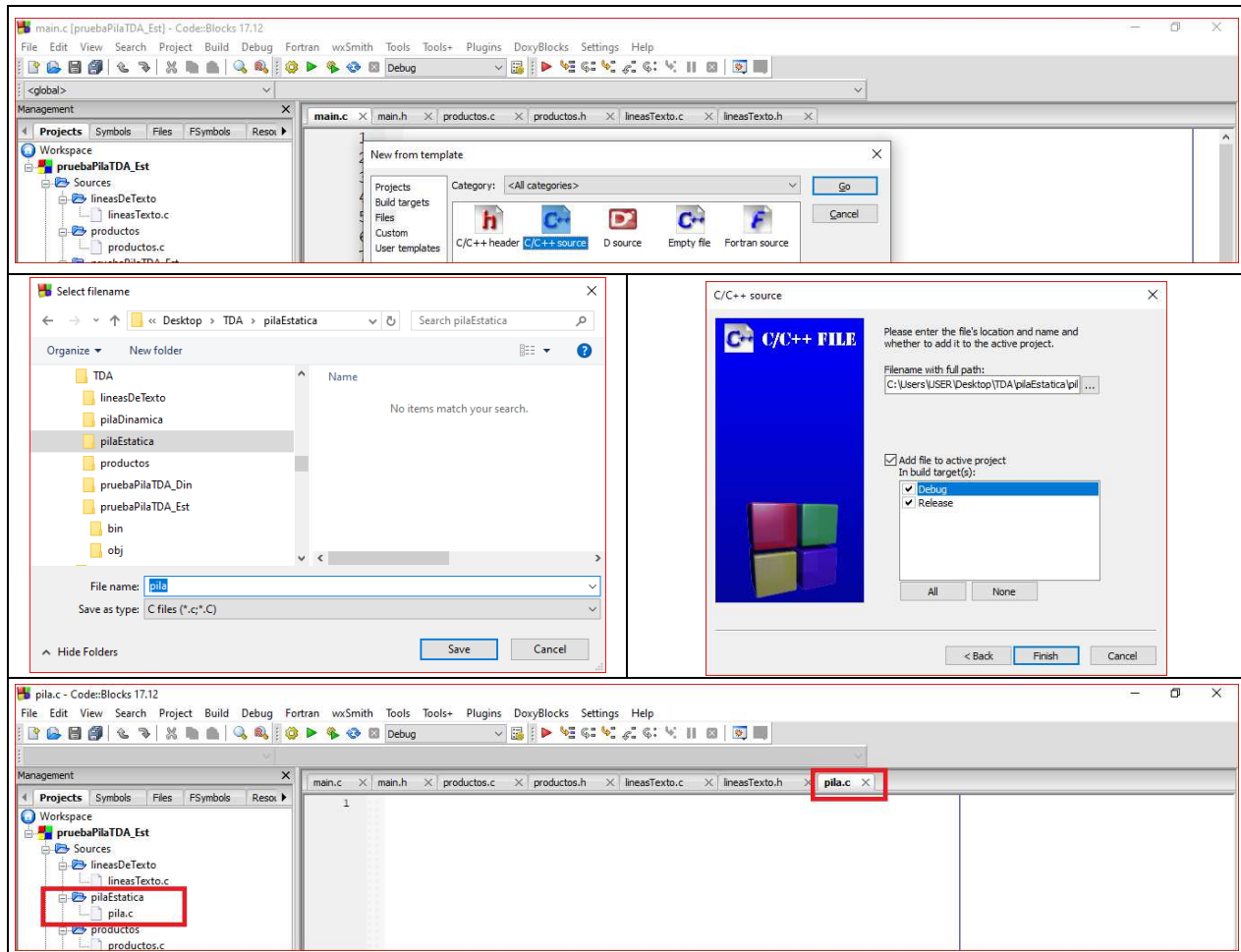
Implementación Estática del TDA PILA

Abra el proyecto "**pruebaPilaTDA_Est.cbp**" haciéndole "doble click" con el mouse . . .

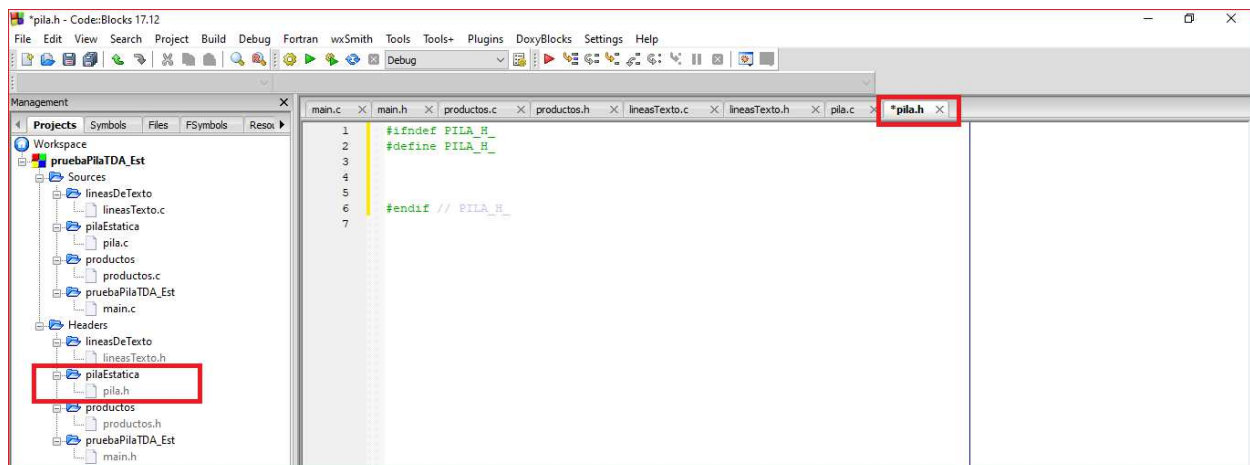


Se acuerda de **[Alt] + [F] - [N] - [F] - [Enter]**, de lo contrario con el menú desplegable **[File] / [New] / [File...]**, procedemos a agregar "**pila.c**" para la Pila Estática . . .





... y con una secuencia similar el de "pila.h" ...





UNLaM

Dto. Ingeniería e Investigaciones Tecnológicas

CodeBlocks 17.12 - *pila.h [pruebaPilaTDA_Est]

File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help

Management

Workspace

pruebaPilaTDA_Est

Sources

linesDeTexto.c

linesTexto.c

pila.c

productos.c

productos.h

pruebaPilaTDA_Est

main.c

Headers

linesDeTexto.h

linesTexto.h

pila.h

productos.h

pruebaPilaTDA_Est

main.h

main.c

main.h

productos.c

productos.h

linesTexto.c

linesTexto.h

pila.c

*pila.h

1 /*
2 * pila.h ESTÁTICA
3 */
4
5 #ifndef PILA_H
6 #define PILA_H
7
8 #include <string.h>
9
10 #define minimo(X , Y) ((X) <= (Y) ? (X) : (Y))
11
12 #define TAM_PILA 340
13
14 typedef struct
15 {
16 char pila[TAM_PILA];
17 unsigned tope;
18 } tPila;
19
20 void crearPila(tPila *p);
21 int pilaLlena(const tPila *p, unsigned cantBytes);
22 int ponerEnPila(tPila *p, const void *d, unsigned cantBytes);
23 int verTope(const tPila *p, void *d, unsigned cantBytes);
24
25

Complete el código correspondiente a "pila.h"

26 int pilaVacía(const tPila *p);
27 int sacarDePila(tPila *p, void *d, unsigned cantBytes);
28 void vaciarPila(tPila *p);
29
30 #endif
31
32

C:\Users\USER\Desktop\TDA\pilaEstática\pila.h C/C++ Windows (CR+LF) WINDOWS-1252 Line 32, Col 1, Pos 795 Insert Modified Read/Write default

CodeBlocks 17.12 - *pila.c [pruebaPilaTDA_Est]

File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help

Management

Workspace

pruebaPilaTDA_Est

Sources

linesDeTexto.c

linesTexto.c

pila.c

productos.c

productos.h

pruebaPilaTDA_Est

main.c

Headers

linesDeTexto.h

linesTexto.h

pila.h

productos.h

pruebaPilaTDA_Est

main.h

main.c

main.h

productos.c

productos.h

linesTexto.c

linesTexto.h

pila.c

*pila.h

1 /*
2 * pila.c ESTÁTICA
3 */
4
5 #include "pila.h"
6
7
8 void crearPila(tPila *p)
9 {
10 p->tope = TAM_PILA;
11 }
12
13 int pilaLlena(const tPila *p, unsigned cantBytes)
14 {
15 return p->tope < cantBytes + sizeof(unsigned);
16 }
17
18 int ponerEnPila(tPila *p, const void *d, unsigned cantBytes)
19 {
20 if(p->tope < cantBytes + sizeof(unsigned))
21 return 0;
22 p->tope -= cantBytes;
23 memcpy(p->pila + p->tope, d, cantBytes);
24 p->tope -= sizeof(unsigned);
25

Complete el código correspondiente a "pila.c"



UNLaM

Dto. Ingeniería e Investigaciones Tecnológicas

```

26     memcpy(p->pila + p->tope, &cantBytes, sizeof(unsigned));
27     return 1;
28 }
29
30 int verTope(const tPila *p, void *d, unsigned cantBytes)
31 {
32     unsigned tamInfo;
33
34     if(p->tope == TAM_PILA)
35         return 0;
36     memcpy(&tamInfo, p->pila + p->tope, sizeof(unsigned));
37     memcpy(d, p->pila + p->tope + sizeof(unsigned),
38           minimo(cantBytes, tamInfo));
39     return 1;
40 }
41
42 int pilaVacía(const tPila *p)
43 {
44     return p->tope == TAM_PILA;
45 }
46
47 int sacarDePila(tPila *p, void *d, unsigned cantBytes)
48 {
49     unsigned tamInfo;
50
51     if(p->tope == TAM_PILA)
52         return 0;
53     memcpy(&tamInfo, p->pila + p->tope, sizeof(unsigned));
54     p->tope += sizeof(unsigned);
55     memcpy(d, p->pila + p->tope, minimo(cantBytes, tamInfo));
56     p->tope += tamInfo;
57     return 1;
58 }
59
60 void vaciarPila(tPila *p)
61 {
62     p->tope = TAM_PILA;
63 }
64
65

```

... y ahora, para que en "main.c" se puedan utilizar las primitivas de pila y el tipo de dato para la pila, vaya a "main.h", y agregue el correspondiente *include* ...

```

1  #ifndef MAIN_H
2  #define MAIN_H
3
4  #include <stdio.h>
5
6
7  #include "../productos/productos.h"
8  #include "../lineasDeTexto/lineasTexto.h"
9  #include
10
11  void pro ..../lineasDeTexto/lineasTexto.h
12
13  void pro ..../productos/productos.h
14  accctrl.h

```



UNLaM

Dto. Ingeniería e Investigaciones Tecnológicas

```

1  #ifndef MAIN_H
2  #define MAIN_H
3
4  #include <stdio.h>
5
6
7  #include "../productos/productos.h"
8  #include "../lineasDeTexto/lineasTexto.h"
9  #include "../pilaEstatica/pila.h"
10
11 void probarIngresarYMostrarProd(void);
12
13 void probarIngresarYMostrarTexto(void);
14
15 void probarPonerYSacarDePila(void);
16
17 #endif // MAIN_H
18
19

```

... además del prototipo de la función con la que se probará el TDA Pila.

Además, en la función "main" queda pendiente invocar a esta función ...

```

1  #include "main.h"
2
3
4  int main()
5  {
6      probarIngresarYMostrarProd();
7
8      probarIngresarYMostrarTexto();
9
10     probarPonerYSacarDePila();
11
12     return 0;
13 }
14
15
16 void probarIngresarYMostrarProd(void)

```

Escriba un desarrollo mínimo para la función ("probarPonerYSacarDePila"), que hará las pruebas de las primitivas (que tan solo manifieste su presencia);: ...

```

56 void probarPonerYSacarDePila(void)
57 {
58     puts("Probando primitivas de pila con productos.\n");
59     "===== \n";
60 }
61

```



UNLaM

Dto. Ingeniería e Investigaciones Tecnológicas

Compile, ejecute, y vea los resultados . . .

```

C:\Users\USER\Desktop\TDA\pruebaPilaTDA_Est\bin\Debug\pruebaPilaTDA_Est.exe
Probando ingresar productos y mostrar productos.
=====
Cod. Produccion del producto
clavoro3/4 Clavo de oro 24 kilates de 3/4 de pulgada
martillo3K Martillo bolita con saca clavos de 3 kilos
alameso1 Alambre de yeso de un milimetro de espesor
rem-vid15 Remache de vidrio de 1,5 milímetros
plom-telgo Plomada de poliestireno expandido
limagoma17 Lima de goma de 17 pulgadas
Se mostraron 6 productos.

Probando ingresar lineas de texto mostrandolas.
=====
"Se necesita un amigo - Fragmento del Poema de Vinicius de Moraes"
""
"Debe tener un ideal, y miedo de perderlo,"
"y en caso de no ser asi,"
"debe sentir el gran vacio que esto deja."
"Tiene que tener resonancias humanas,"
"su principal objetivo debe ser el del amigo."
"Debe sentir pena por las personas tristes"
"y comprender el inmenso vacio de los solitarios."
"Se busca un amigo para gustar"
"de los mismos gustos,"
"que se conmueva cuando es tratado de amigo."
""
Se mostraron 13 lineas de texto.

Probando primitivas de pila con productos.
=====

```

Y ahora, codifique para utilizar y probar todas las primitivas de Pila, poniendo en la misma productos o líneas de texto . . .

```

*main.c [pruebaPilaTDA_Est] - Code::Blocks 17.12
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
<global>
Management
Workspace
pruebaPilaTDA_Est
Sources
lineasDeTexto
lineasTexto.c
pilaEstadistica
pila.c
productos
productos.c
pruebaPilaTDA_Est
main.c
Headers
lineasDeTexto
lineasTexto.h
pilaEstadistica
pila.h
productos
productos.h
pruebaPilaTDA_Est
main.h

30
31 void probarIngresarYMostrarTexto(void)
32 {
33     char linea[90];
34     int cant = 0;
35
36     puts("Probando ingresar lineas de texto mostrandolas.\n"
37         "===== ");
38     while(ingresarTexto(linea, sizeof(linea)))
39     {
40         cant++;
41         printf("\n%s", linea);
42     }
43     fprintf(stdout, "Se mostraron %d lineas de texto.\n", cant);
44 }
45
46
47 /**
48  ** DE NINGUNA MANERA ES ADMISIBLE HACER UNA FUNCION "MONOLITICA" TAN LARGA.
49  ** USTED DEBERIA DIVIDIRLA EN VARIAS FUNCIONES.
50  ** EN CADA FUNCION PROBAR UNA O DOS PRIMITIVAS.
51  ** A ESAS FUNCIONES DEBERIA PASARLES LA PILA (por puntero).
52  ** (VER AL FINAL)
53  **/
54 void probarPonerYSacarDePila(void)

```




UNLaM

Dto. Ingeniería e Investigaciones Tecnológicas

```

55  {
56      tProd    prod,
57      |        otro;
58      char     linea[70];
59      tPila    pila;
60      int      cant;
61
62      crearPila(&pila);
63
64      puts("Probando primitivas de pila con productos.\n"
65          | "===== \n"
66          | "Probando pila llena y poner en pila.");
67      mostrarProducto(NULL);
68      cant = 0;
69      while(!pilaLlena(&pila, sizeof(prod)) && ingresarProducto(&prod))
70      {
71          if(!ponerEnPila(&pila, &prod, sizeof(prod)))
72          {
73              fprintf(stderr, "ERROR - inesperado - pila llena.\n");
74              puts("no se pudo cargar la informacion y"
75                  | " habria que tomar alguna decision drastica.");
76          }
77          mostrarProducto(&prod);
78          cant++;
79      }
80      printf("se pusieron %d productos en la pila.\n\n", cant);
81      puts("Probando ver el tope de la pila.");
82      if(verTope(&pila, &otro, sizeof(otro)))
83      {
84          mostrarProducto(NULL);
85          mostrarProducto(&otro);
86      }
87      else
88          puts("La pila estaba vacia.");
89      puts("");
90      cant -= 2;
91      printf("Probando pila vacia y sacar de pila %d productos (mostrandolos.\n",
92          | cant);
93      if(pilaVacía(&pila))
94          puts("La pila esta vacia.");
95      else
96          mostrarProducto(NULL);
97      while(cant > 0 && sacarDePila(&pila, &prod, sizeof(prod)))
98      {
99          cant--;
100          mostrarProducto(&prod);
101      }
102      puts("");
103      puts("Probando ver el tope de la pila.");
104      if(verTope(&pila, &otro, sizeof(otro)))
105      {
106          puts("La pila no quedo vacia - en el tope hay ...");
107          mostrarProducto(NULL);
108          mostrarProducto(&otro);
109      }
110      else
111          puts("La pila esta vacia.");
112      puts("");
113      puts("Probando vaciar pila y pila vacia.");
114      vaciarPila(&pila);
115      if(!pilaVacía(&pila))
116          fprintf(stderr, "ERROR - la pila debia estar vacia\n\n");
117      else
118          printf("Vaciar pila funciona!\n\n");
119      puts("");
120
121      puts("Probando primitivas de pila con lineas de texto.\n"
122          | "===== \n"
123          | "Probando pila llena y poner en pila.");
124      cant = 0;
125      while(!pilaLlena(&pila, sizeof(prod)) && ingresarTexto(linea, sizeof(linea)))
126      {

```




UNLaM

Dto. Ingeniería e Investigaciones Tecnológicas

```

Workspace
├── pruebaPilaTDA_Est
│   ├── Sources
│   │   ├── lineaDeTexto
│   │   │   ├── lineaDeTexto.c
│   │   │   └── lineaDeTexto.h
│   │   ├── pilaEstatica
│   │   │   ├── pila.c
│   │   │   └── pila.h
│   │   ├── productos
│   │   │   ├── productos.c
│   │   │   └── productos.h
│   │   ├── pruebaPilaTDA_Est
│   │   │   ├── main.c
│   │   │   └── main.h
│   └── Headers
└── pruebaPilaTDA_Est

127         if(!ponerEnPila(&pila, linea, strlen(linea) + 1))
128         {
129             fprintf(stderr, "ERROR - inesperado - pila llena.\n");
130             puts("no se pudo cargar la informacion y"
131                 " habria que tomar alguna decision drastica.");
132         }
133         printf("\n%s\n", linea);
134         cant++;
135     }
136     printf("se pusieron %d lineas de texto en la pila.\n\n", cant);
137     printf("Probando sacar de pila con las lineas de texto.\n");
138     cant = 0;
139     while(sacarDePila(&pila, linea, sizeof(linea)))
140     {
141         cant++;
142         printf("\n%s\n", linea);
143     }
144     printf("Se sacaron y mostraron %d lineas de texto\n\n", cant);
145     puts("ATENCION: se mostro el uso de una pila, en la que se apilaron"
146         " productos, se pro\n"
147         "baron todas las primitivas. Una vez que se la dejo vacia "
148         "se apilaron lineas\n"
149         "de texto y luego se desapilaron. Lo mas importante de esto "
150         "no es utilizar la\n");

127         if(!ponerEnPila(&pila, linea, strlen(linea) + 1))
128         {
129             fprintf(stderr, "ERROR - inesperado - pila llena.\n");
130             puts("no se pudo cargar la informacion y"
131                 " habria que tomar alguna decision drastica.");
132         }
133         printf("\n%s\n", linea);
134         cant++;
135     }
136     printf("se pusieron %d lineas de texto en la pila.\n\n", cant);
137     printf("Probando sacar de pila con las lineas de texto.\n");
138     cant = 0;
139     while(sacarDePila(&pila, linea, sizeof(linea)))
140     {
141         cant++;
142         printf("\n%s\n", linea);
143     }
144     printf("Se sacaron y mostraron %d lineas de texto\n\n", cant);
145     puts("ATENCION: se mostro el uso de una pila, en la que se apilaron"
146         " productos, se pro\n"
147         "baron todas las primitivas. Una vez que se la dejo vacia "
148         "se apilaron lineas\n"
149         "de texto y luego se desapilaron. Lo mas importante de esto "
150         "no es utilizar la\n");

151         "misma pila, lo que mas importa es que con las mismas primiti"
152         "vas se pueden api-\n"
153         "lar distintos tipos de objetos, incluyendo lineas de texto d"
154         "e distinto tamano.\n");
155     }
156 }
157
158 /**
159 int _probarLlenaYApilar(tPila *pila)
160 {
161     tProd prod;
162     int cant = 0;
163
164     puts("Probando pila llena y poner en pila.");
165     mostrarProducto(NULL);
166     while(!pilaLlena(pila, sizeof(prod)) && ingresarProducto(&prod))
167     {
168         if(!ponerEnPila(pila, &prod, sizeof(prod)))
169         {
170             fprintf(stderr, "ERROR - inesperado - pila llena.\n");
171             puts("no se pudo cargar la informacion y"
172                 " habria que tomar alguna decision drastica.");
173         }
174         mostrarProducto(&prod);

```



UNLaM

Dto. Ingeniería e Investigaciones Tecnológicas

```

Workspace
├── pruebaPilaTDA_Est
│   ├── Sources
│   │   ├── lineaDeTexto
│   │   │   ├── lineaDeTexto.c
│   │   │   └── lineaDeTexto.h
│   │   ├── pilaEstatica
│   │   │   ├── pila.c
│   │   │   └── pila.h
│   │   ├── productos
│   │   │   ├── productos.c
│   │   │   └── productos.h
│   │   ├── pruebaPilaTDA_Est
│   │   │   ├── main.c
│   │   │   └── main.h
│   └── Headers
│       ├── lineaDeTexto
│       │   ├── lineaDeTexto.c
│       │   └── lineaDeTexto.h
│       ├── pilaEstatica
│       │   ├── pila.c
│       │   └── pila.h
│       ├── productos
│       │   ├── productos.c
│       │   └── productos.h
│       └── pruebaPilaTDA_Est
│           ├── main.c
│           └── main.h
└── Headers
    ├── lineaDeTexto
    │   ├── lineaDeTexto.c
    │   └── lineaDeTexto.h
    ├── pilaEstatica
    │   ├── pila.c
    │   └── pila.h
    ├── productos
    │   ├── productos.c
    │   └── productos.h
    └── pruebaPilaTDA_Est
        ├── main.c
        └── main.h

```

```

175         cant++;
176     }
177     return cant;
178 }
179
180 void _probarVerTope(tPila *pila)
181 {
182     tProd prod;
183
184     puts("Probando ver el tope de la pila.");
185     if(verTope(pila, &prod, sizeof(prod)))
186     {
187         mostrarProducto(NULL);
188         mostrarProducto(&prod);
189     }
190     else
191         puts("La pila estaba vacia.");
192     puts("");
193 }
194
195 void _probarVaciarYDesapilarN(tPila *pila, int canti)
196 {
197     tProd prod;
198
199     printf("Probando pila vacia y sacar de pila %d productos (mostrandolos.\n",
200           canti);
201     if(pilaVaciar(pila))
202         puts("La pila esta vacia.");
203     else
204         mostrarProducto(NULL);
205     while(canti > 0 && sacarDePila(pila, &prod, sizeof(prod)))
206     {
207         canti--;
208         mostrarProducto(&prod);
209     }
210     puts("");
211 }
212
213 void probarPonerYSacarDePila_2(void)
214 {
215     tPila pila;
216     int cant;
217
218     crearPila(&pila);
219
220     puts("Probando primitivas de pila con productos.\n");
221     printf("===== \n");
222     cant = _probarLlenarYApilar(&pila);
223
224     puts("Probando primitivas de pila con lineas de texto.\n");
225     printf("===== \n");
226     cant = 0;
227     while(!pilaLlena(&pila, sizeof(prod)) && ingresarTexto(linea, sizeof(linea)))
228     {
229         if(!ponerEnPila(&pila, linea, strlen(linea) + 1))
230         {
231             fprintf(stderr, "ERROR - inesperado - pila llena.\n");
232             puts("no se pudo cargar la informacion y\n");
233             puts("habria que tomar alguna decision drastica.");
234         }
235         printf("\n%s\n", linea);
236         cant++;
237     }
238     printf("se pusieron %d lineas de texto en la pila.\n", cant);
239     printf("Probando sacar de pila con las lineas de texto.\n");
240     cant = 0;
241     while(sacarDePila(&pila, linea, sizeof(linea)))
242     {
243         cant++;
244         printf("\n%s\n", linea);
245     }
246 }

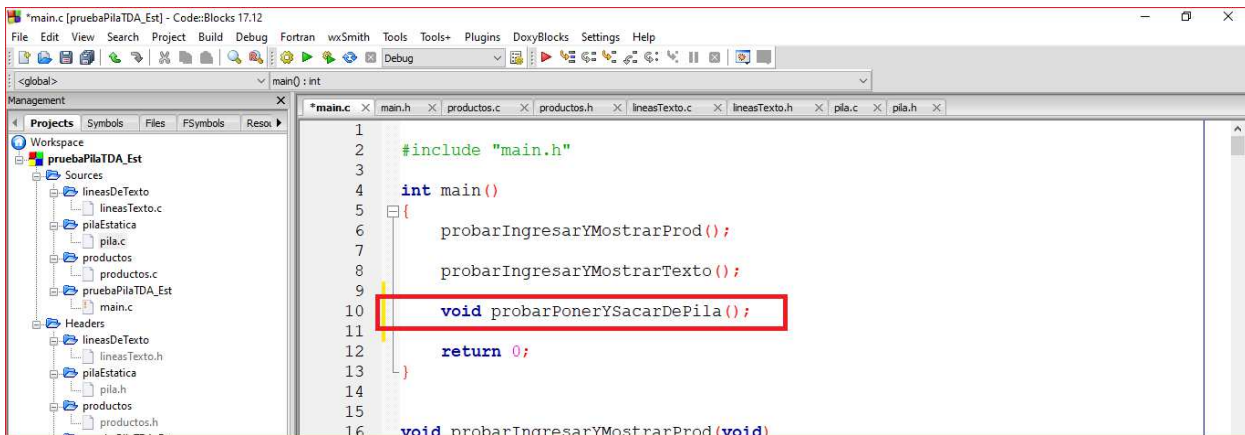
```

```

271     printf("Se sacaron y mostraron %d lineas de texto\n\n", cant);
272     puts("ATENCION: se mostro el uso de una pila, en la que se apilaron"
273         " productos, se pro\n"
274         "baron todas las primitivas. Una vez que se la dejo vacia "
275         "se apilaron lineas\n"
276         "de texto y luego se desapilaron. Lo mas importante de esto "
277         "no es utilizar la\n"
278         "misma pila, lo que mas importa es que con las mismas primiti"
279         "vas se pueden api-\n"
280         "lar distintos tipos de objetos, incluyendo lineas de texto d"
281         "e distinto tamano.\n");
282 }
283 /**
284

```

Y finalmente en la función "**main**", quedaba pendiente invocar a esta función . . .



```

1
2  #include "main.h"
3
4  int main()
5  {
6      probarIngresarYMostrarProd();
7
8      probarIngresarYMostrarTexto();
9
10     void probarPonerYSacarDePila();
11
12     return 0;
13 }
14
15
16 void probarIngresarYMostrarProd(void)

```

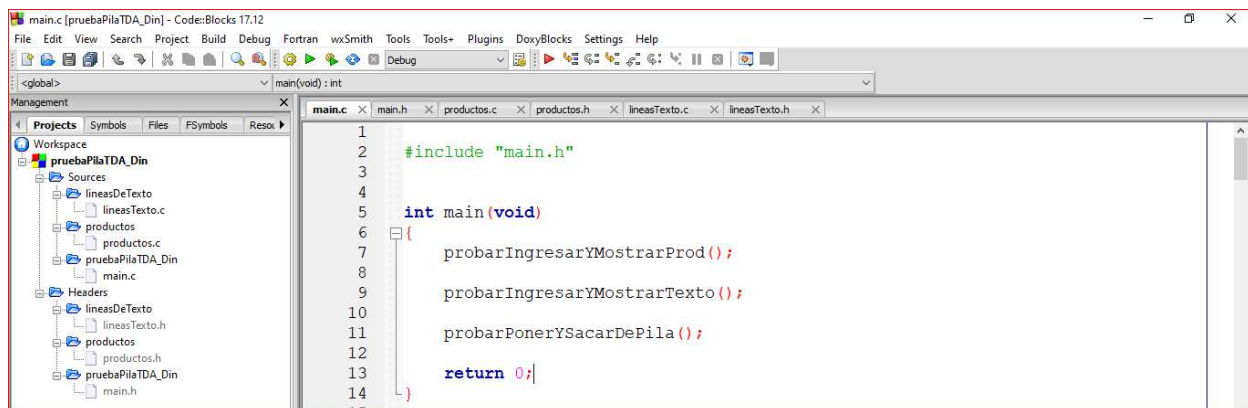
En el ejemplo anterior en una sola función se prueban todas las primitivas.

Esto resulta en una función monolítica con muchísimas líneas de código en la que se hace una prueba exhaustiva. Le queda como tarea hacer que esa función invoque a funciones que prueben unas pocas primitivas, lo que le dará la libertad de ir las probando de a poco. El modo sugerido le va a resultar, espero, más que obvio para invocar funciones de servicio (no tienen su prototipo *publicado* en "**main.h**"; y habitualmente por pauta de estilo el primer carácter del identificador de esas funciones es el "_" (guion bajo). Dado que no hay prototipos de estas, sus desarrollos deben preceder a la función que las invoca.

Implementación Dinámica del TDA PILA

Dado que en el proyecto de prueba del TDA Pila con implementación estática ya ha generado el código correspondiente, proceda a copiar "**main.c**" de la carpeta "**pruebaPilaTDA_Est**" a la carpeta "**pruebaPilaTDA_Din**".

Abra el proyecto . . .



. . . y proceda con "**pila.c**" y "**pila.h**". ¿Recuerda los golpes de tecla [Alt] + [F] - [N] - [F] - [Enter]? Proceda a crear ambos archivos fuente en la carpeta "**pilaDinamica**" y a codificarlos.



UNLaM

Dto. Ingeniería e Investigaciones Tecnológicas

Two screenshots of the Code::Blocks 17.12 IDE showing the implementation of a stack (Pila) data structure.

Top Screenshot: *pila.h - Code::Blocks 17.12

```

1  #ifndef PILA_H
2  #define PILA_H
3
4  #include <stdlib.h>
5  #include <string.h>
6
7
8  typedef struct sNodo
9  {
10     void          *info;
11     unsigned      tamInfo;
12     struct sNodo  *sig;
13 } tNodo;
14 typedef tNodo *tPila;
15
16 void crearPila(tPila *p);
17 int  pilaLlena(const tPila *p, unsigned cantBytes);
18 int  ponerEnPila(tPila *p, const void *d, unsigned cantBytes);
19 int  verTope(const tPila *p, void *d, unsigned cantBytes);
20 int  pilaVacía(const tPila *p);
21 int  sacarDePila(tPila *p, void *d, unsigned cantBytes);
22 void vaciarPila(tPila *p);
23
24 #endif
25

```

Bottom Screenshot: *pila.c [pruebaPilaTDA_Din] - Code::Blocks 17.12

```

1  #include "pila.h"
2
3  #define minimo( X , Y )    ( ( X ) <= ( Y ) ? ( X ) : ( Y ) )
4
5  void crearPila(tPila *p)
6  {
7      *p = NULL;
8  }
9
10 int  pilaLlena(const tPila *p, unsigned cantBytes)
11 {
12     tNodo  *aux = (tNodo *)malloc(sizeof(tNodo));
13     void    *info = malloc(cantBytes);
14
15     free(aux);
16     free(info);
17     return aux == NULL || info == NULL;
18 }
19
20 int  ponerEnPila(tPila *p, const void *d, unsigned cantBytes)
21 {
22     tNodo *nue;
23
24 }
25

```




UNLaM

Dto. Ingeniería e Investigaciones Tecnológicas

```

26
27     if((nue = (tNodo *)malloc(sizeof(tNodo))) == NULL ||
28         (nue->info = malloc(cantBytes)) == NULL)
29     {
30         free(nue);
31         return 0;
32     }
33     memcpy(nue->info, d, cantBytes);
34     nue->tamInfo = cantBytes;
35     nue->sig = *p;
36     *p = nue;
37     return 1;
38 }
39
40 int verTope(const tPila *p, void *d, unsigned cantBytes)
41 {
42     if(*p == NULL)
43         return 0;
44     memcpy(d, (*p)->info, minimo(cantBytes, (*p)->tamInfo));
45     return 1;
46 }
47
48 int pilaVacía(const tPila *p)
49 {
50     return *p == NULL;
51 }
52
53 int sacarDePila(tPila *p, void *d, unsigned cantBytes)
54 {
55     tNodo *aux = *p;
56
57     if(aux == NULL)
58         return 0;
59     *p = aux->sig;
60     memcpy(d, aux->info, minimo(cantBytes, aux->tamInfo));
61     free(aux->info);
62     free(aux);
63     return 1;
64 }
65
66 void vaciarPila(tPila *p)
67 {
68     while(*p)
69     {
70         tNodo *aux = *p;
71
72         *p = aux->sig;
73         free(aux->info);
74         free(aux);
75     }
76 }
77
78

```

Vaya a "**main.c**" y seleccione para copiarlo, la definición del encabezado del desarrollo de la función ("**void probarPonerYSacarDePila(void)**") y *pegue* en "**main.h**" la declaración (prototipo), de la función y además agregue el *include* a "**pila.h**".



UNLaM

Dto. Ingeniería e Investigaciones Tecnológicas

```

56 void probarPonerYSacarDePila(void)
57 {
58     tProd prod,
59     otro;
60     char linea[70];
61     tPila pila;
62     int cant;
63
1  #ifndef MAIN_H
2  #define MAIN_H
3
4  #include <stdio.h>
5
6
7  #include "../productos/productos.h"
8  #include "../lineasDeTexto/lineasTexto.h"
9  #include "../pilaDinamica/pila.h"
10
11
12 void probarIngresarYMostrarProd(void);
13
14 void probarIngresarYMostrarTexto(void);
15
16 void probarPonerYSacarDePila(void);
17
18
19 #endif // MAIN_H
20
21

```

Si todo anduvo bien, este proyecto compilará y ejecutará *mejor* que el otro, dado que no se llenará la pila.

Pruébalo.

Note un pequeño detalle. En "pila.h" de "pilaEstática" se escribió el macro reemplazo que determina el mínimo entre dos valores. En cambio, en "pilaDinamica" se lo escribió en el lugar que se lo necesita, en "pila.c".

Cuando se declaran macro reemplazos en archivos ".h", deberían estar dentro de una directiva de compilación condicional, por ejemplo:

```

#ifndef minimo
#define minimo( X , Y )    ( ( X ) <= ( Y ) ? ( X ) : ( Y ) )
#endif //minimo

```

Averigüe por qué.



UNLaM

Dto. Ingeniería e Investigaciones Tecnológicas

OBSERVACIÓN: cuando termine de probar sus proyectos si los quiere trasladar a otra computadora o mandarlos por correo, todo lo que tiene que llevar son los archivos ".c", ".h", ".cbp", y ".layout" dentro de sus respectivas *carpetas* (junto con la *carpeta* de la cual dependen, en este caso TDA). Elimine cuando termina de trabajar cualquier otro archivo como los ".depend" además de las *carpetas* "bin" y "obj".

Si lo quiere mandar por correo, debería comprimir la *carpeta* TDA completa. Tenga en cuenta que los correos le rechazarán el envío de archivos ejecutables aun cuando estén comprimidos.

Si lo quiere pasar a otra *carpeta copie* (o *corte*), todas las *carpetas* dentro de la *carpeta* TDA y péguelas en la *carpeta* destino.

Si la quiere renombrar, hágalo sin mayor preocupación.

¡Bueno, como habrá visto, nos hemos ido un poco más allá de los límites de la materia!, pero son esos detalles los que nos enriquecen.