10.2. Un programa a partir de varios fuentes en C++

Como hemos dicho, un motivo habitual para descomponer un proyecto en varias clases es poder repartir trabajo. Por eso, vamos a ver primero cómo podríamos "partir un programa en dos trozos", todavía sin emplear clases.

Vamos a partir de un programa que contenga dos funcione: una función "uno", que escribirá "uno" en pantalla, y otra función "dos", que escriba el correspondiente texto en pantalla:

// Introduccion a C++, Nacho Cabanes

// Ejemplo 10.01:

// Repartir en varios fuentes: previo

#include

using namespace std;

void uno()

{

cout << "uno" << endl;

}

void uno()

{

cout << "dos" << endl;

}

int main()

{

uno();

dos();

return 0;

}

Ahora vamos a repartirlo en dos fuentes distintos, de modo que una persona se pudiera encargar de las mejoras y cambios que necesitara la función "uno", mientras que otra persona mantuviera la función "dos".

Si usamos un IDE (entorno de desarrollo integrado), deberíamos **crear un proyecto**, para indicar que nuestro programa va a estar formado por varios fuentes distintos. Si compilamos "desde línea de comandos", usaremos nuestro editor para crear cada uno de los fuentes por separado y luego lanzaremos el compilador, indicando todos los fuentes que deberá enlazar:

g++ ej1002.cpp ej1002uno.cpp ej1002dos.cpp -o ej1002

Donde "ej1002.cpp" sería el nuevo programa principal, que sólo contendría "main", y ni siquiera necesitará incluir "iostream", porque no acceder a pantalla directamente. Podríamos esperar que fuera así:

// Introducción a C++, Nacho Cabanes

// Ejemplo 10.02:

// Repartir en varios fuentes: principal (primer intento)

int main()

{

uno();

dos();

return 0;

}

Pero eso no va a compilar correctamente, porque no sabe qué son "uno" y "dos", y deberemos indicárselo (lo haremos en un instante).

El primero de esos subprogramas, el que contendría la función "uno", sería:

// Introducción a C++, Nacho Cabanes

// Ejemplo 10.02uno:

// Repartir en varios fuentes: función "uno"

#include

using namespace std;

void uno()

{

cout << "uno" << endl;

}

Y el segundo, con la función "dos", sería muy similar:

// Introducción a C++, Nacho Cabanes

// Ejemplo 10.02dos:

// Repartir en varios fuentes: funcion "dos"

#include

using namespace std;

void dos()

{

cout << "dos" << endl;

}

Pero al intentar compilar obtenemos el mensaje de error que nos dice que le estamos hablando de un tal "uno" y de un tal "dos" que no conoce:

g++ ej1002.cpp ej1002uno.cpp ej1002dos.cpp -o ej1002

ej1002.cpp: In function â€˜int main()â€™:

ej1002.cpp:7:9: error: â€˜unoâ€™ was not declared in this scope

uno();

^

ej1002.cpp:8:9: error: â€˜dosâ€™ was not declared in this scope

dos();

^

Una primera solución, que funciona pero que todavía no es la ideal, consiste en incluir los "**prototipos**" de las funciones (los nombres y parámetros, sin detalles cobre cómo son internamente) antes de "main":

// Introducción a C++, Nacho Cabanes

// Ejemplo 10.02b:

// Repartir en varios fuentes: principal (segundo intento)

// Prototipos de las funciones

void uno();

void dos();

// Y cuerpo del programa

int main()

{

uno();

dos();

return 0;

}

Esta solución no es perfecta, porque supone que dentro del fichero que contiene "main" haya detalles de otras funciones, y eso es algo que deberíamos evitar, porque estamos aumentando (sin necesidad) el "**acoplamiento**" entre ficheros: unos dependen de los detalles internos de otros. No es demasiado grave al tener sólo dos funciones, pero podría serlo en un fuente real, en el que hubiera muchas funciones, y hubiera que modificar el fichero que contiene "main" (y quizá también otros muchos ficheros) cada vez que cambiemos un detalle en uno de los otros subprogramas.

La alternativa es crear "**ficheros de cabecera**", que recopilarán todos esos prototipos de funciones, y que podremos incluir desde los ficheros que necesiten utilizar esas funciones. Una primera aproximación, todavía no la correcta, sería crear un único fichero de cabecera, que recopilara todos esos prototipos del función que antes estaban en el mismo fichero que "main":

// Introducción a C++, Nacho Cabanes

// Ejemplo 10.02c (cabecera): ej1002c.h

// Repartir en varios fuentes: principal (tercer intento)

// Prototipos de las funciones

void uno();

void dos();

Y el programa principal que lo usaría sería así:

// Introducción a C++, Nacho Cabanes

// Ejemplo 10.02c:

// Repartir en varios fuentes: principal (tercer intento)

#include "ej1002c.h"

int main()

{

uno();

dos();

return 0;

}

**Nota importante**: los ficheros de cabecera, por convenio, se suelen guardar con un nombre terminado en ".h", siguiendo la nomenclatura propuesta para el lenguaje C, o en ".hpp" o ".hh", nomenclaturas que algunos prefieren para los ficheros de cabecera que corresponden a programas en lenguaje C++. Pero es un simple convenio, el compilador se comportará correctamente aunque el nombre del fichero siga una política distinta.

Pero la alternativa "buena", que supone más versatilidad a cambio de escribir más, es crear un fichero de cabecera para cada fichero auxiliar, de modo que programa principal quedaría:

// Introducción a C++, Nacho Cabanes

// Ejemplo 10.02d:

// Repartir en varios fuentes: principal (definitivo)

#include "ej1002uno.h"

#include "ej1002dos.h"

int main()

{

uno();

dos();

return 0;

}

Y tendríamos un fichero de cabecera para el primero fichero fuente auxiliar:

// Introducción a C++, Nacho Cabanes

// Cabecera ej1002uno.h

// Repartir en varios fuentes: uno (cabecera)

void uno();

Y otro para el segundo fichero fuente auxiliar:

// Introducción a C++, Nacho Cabanes

// Cabecera ej1002dos.h

// Repartir en varios fuentes: dos (cabecera)

void dos();

A la hora de compilar el programa, no es necesario indicar los ficheros de cabecera, que se incluyen automáticamente: