Programas de multiples archivos de código

Cualquier programa útil excede la cantidad de líneas de código que es cómodo leer en un sólo archivo. Por eso, en general es práctico separar el código en varios archivos, siguiendo algún patrón (por ejemplo, podemos ordenar por área temática de las rutinas).

Vamos a ver el ejemplo más sencillo. El siguiente código corresponde a un archivo `main.cpp` [1]

```
#include <iostream>

using namespace std;

void hello();

int main() {
        hello();
        return 0;
}
```

Este código ejecuta una sola instrucción: llama a la rutina `hello`. Como pueden ver, en este archivo no está la implementación de la rutina `hello`, solamente podemos ver el formato de la rutina (en este caso vemos que es de tipo `void` y que no lleva argumentos), es decir la declaración de la rutina.

La implementación de la rutina `hello` está escrita en otro archivo, llamado `hello.cpp`, con el siguiente contenido:

```
#include<iostream>

using namespace std;

void hello() {
        cout << "hello world!" << endl;
}
```

Para compilar el código completo debemos tener ambos archivos en el mismo directorio y ejecutar:

```
g++ main.cpp hello.cpp
```

Tengan en cuenta que para que todo funcione bien, en el archivo `hello.cpp` tuvimos que incluir todas las dependencias que usa la rutina `hello` (los `include` y el `namespace`).

[1] En general se acostumbra que el archivo que contiene a la rutina `main` se llame `main.cpp`, y el resto de los archivos se nombran de acuerdo al contenido que tienen.

La forma correcta de compilar

La instrucción para compilar los dos archivos se puede desglosar en instrucciones que procesan cada archivo por separado (generando archivos en estado de compilación intermedia llamados *objects*), y una instrucción final de compilación (que une los archivos). Los pasos de compilación son los siguientes:

`g++ -c hello.cpp`
: Procesa el archivo `hello.cpp` y genera el objeto `hello.o`

`g++ -c main.cpp`
: Procesa el archivo `main.cpp` y genera el objeto `main.o`

`g++ main.o hello.o`
: Compila los objetos `main.o` y `hello.o` y genera el ejecutable final. Este es el paso de compilación en que pueden aparecer problemas de vinculación entre ambos archivos (es decir, los códigos de ambos archivos por separado pueden ser correctos, pero la forma en que los vinculé puede no serlo).

Estos tres pasos de compilación son equivalentes a la línea única de compilación de la sección anterior.

Pero por qué podemos querer usar tres comandos de compilación en lugar de uno?

Si nuestro programa está compuesto de archivos grandes, cuya compilación lleva mucho tiempo (un programa grande puede tardar horas en compilar), seguramente no vamos a querer recompilar todo el código cada vez que modifiquemos una línea. En ese caso, la compilación en varios pasos nos permite recompilar solamente los archivos que hayamos modificado desde la última compilación. En nuestro código de ejemplo, si modificamos el contenido del archivo `hello.cpp`, solamente tenemos que volver a generar `hello.o` y vincularlo con `main.o` (no es necesario volver a generar `main.o`):

```
g++ -c hello.cpp
g++ main.o hello.o
```

Todo muy lindo, pero nadie tiene ganas de fijarse qué archivos modificó al momento de recompilar.

Tal cual, por eso se inventó `Makefile`. Un `Makefile` es un pequeño archivo (a veces no tan pequeño) que tiene adentro la información de cómo se compila nuestro código[2]. Una vez que escribimos correctamente el contenido del `Makefile`, lo colocamos en el mismo directorio que nuestro código, y a partir de ese momento la compilación se realiza simplemente ejecutando el comando:

```
make
```

que interpreta el contenido de `Makefile` y ejecuta los comandos de compilación necesarios. Adicionalmente, si el `Makefile` está correctamente escrito, `make` se

[2] `Makefile` no es un script asociado a `C++`, se puede usar para compilar códigos de cualquier lenguaje (incluido LaTeX).

ocupa automáticamente de ver qué archivos de código se modificaron desde la última compilación, y *recompila únicamente los archivos necesarios*.

El contenido de un archivo `Makefile` sencillo es simplemente una lista de los archivos que se generarán durante la compilación, junto con sus dependencias y las lineas de compilación correspondientes, en el siguiente formato:

```
<archivo que se genera>: <lista de archivos de los que depende>
        <tab obligatorio> <comando de compilacion>
```

En nuestro ejemplo sencillo de los códigos `main.cpp` y `hello.cpp`, el `Makefile` podría ser:

```
programa.x: main.o hello.o
        g++ main.o hello.o -o programa.x

main.o: main.cpp
        g++ -c main.cpp

hello.o: hello.cpp
        g++ -c hello.cpp
```

`programa.x: main.o hello.o`
`g++ main.o hello.o -o programa.x`
El ejecutable del programa se llamará `programa.x`, y va a necesitar previamente de la existencia de dos archivos: `main.o` y `hello.o`.
La linea de compilación es la que genera el ejecutable.

`main.o: main.cpp`
`g++ -c main.cpp`
Para generar el objeto `main.o` se necesita `main.cpp`. Esto le indica a `make` que debe generar nuevamente `main.o` si `main.cpp` se modificó desde la última compilación.
La linea de compilación es la que genera el objeto `main.o`.

`hello.o: hello.cpp`
`g++ -c hello.cpp`
Para generar el objeto `hello.o` se necesita `hello.cpp`. Esto le indica a `make` que debe generar nuevamente `hello.o` si `hello.cpp` se modificó desde la última compilación.
La linea de compilación es la que genera el objeto `hello.o`.