# 1. HERRAMIENTAS DE AYUDA A LA CREACIÓN DE PROGRAMAS:

1.1. Compilación separada (make)
1.2. Entornos de desarrollo
1.3. Bibliotecas de programas: estáticas y dinámicas
1.4. Herramientas de documentación
1.5. Análisis de rendimiento de programas (profiling)
1.6. Control de versiones (CVS)



## Compilación separada

- Un programa escrito en *C/C++* normalmente está compuesto por varios archivos.
- Los archivos se van modificando según se va elaborando o modificando el programa.
- Cada vez que se modifica algún archivo del programa, éste debe recompilarse para generar un ejecutable del programa actualizado.
- La modificación de un archivo del programa no implica que se deban recompilar todos los archivos del programa, sólo los que se ven afectados por la modificación (dependencias).

## Ejemplo: Clase “fecha” (interfaz)

fecha.h

```
class fecha
{
   private:
        unsigned dia;
        unsigned mes;
        unsigned anyo;
   public:
        fecha crearFecha (unsigned, unsigned, unsigned);
        void leerFecha (istream&);
        void escribirFecha (ostream&);
        bool igualFecha (fecha);
        bool menorFecha (fecha);
        bool mayorFecha (fecha);
};
```



## Ejemplo: Clase “fecha” (implementación)

fecha.cpp

```
#include <fstream.h>
#include "fecha.h"

fecha fecha::crearFecha (unsigned, unsigned, unsigned)
   { //implementación }
void fecha::leerFecha (istream&);
   { //implementación }
void fecha::escribirFecha (ostream&);
   { //implementación }
bool fecha::igualFecha (fecha);
   { //implementación }
bool fecha::menorFecha (fecha);
   { //implementación }
bool fecha::mayorFecha (fecha);
   { //implementación }
```

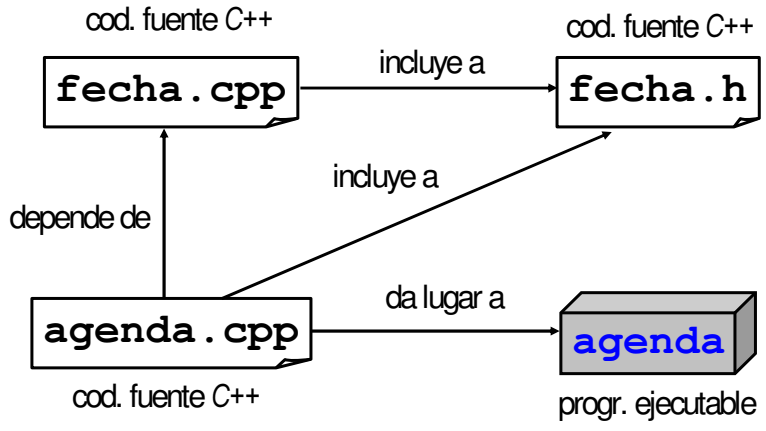
## Ejemplo: Clase “fecha” (utilización)

agenda.cpp

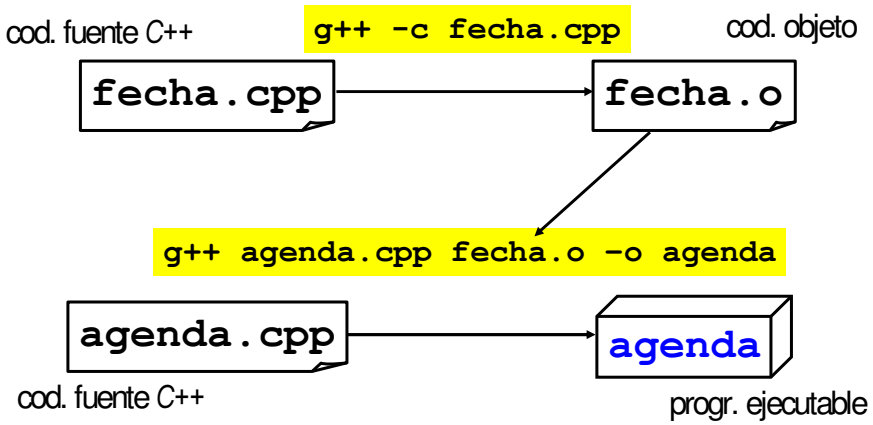
```
#include <fstream.h>
#include <string>
#include "fecha.h"

int main()
{
    struct aniversario
    {
         string persona;
         fecha f;
    };
//...
    for (i = 0; i < 10; i++)
    {
         cout << "Nombre: ";
         cin >> recordar[i].persona;
         cout << "Fecha: ";
         recordar[i].f.leerFecha(cin);
    } //...
```



## Ejemplo: Dependencias para generar ejecutable




## Ejemplo: Compilación






## make

- La utilidad `make` determina automáticamente qué archivos del programa deben ser recompilados y las órdenes que se deben utilizar para ello.
- En programas muy grandes nos ayuda a mantener el ejecutable totalmente actualizado.
- No debemos recordar las dependencias entre archivos del programa ni las órdenes necesarias para compilar cada parte del programa.

## Utilización de `make`

- Para usar el `make` debemos escribir un fichero llamado ***`Makefile`***. Este fichero describe las relaciones entre ficheros y las ordenes que se deben ejecutar con cada archivo.
- Si en un directorio existe el fichero ***`Makefile`***, para ejecutar la utilidad `make` simplemente debemos escribir en la línea de órdenes:

  *prompt$* make
- Con esta orden `make` busca el fichero ***`Makefile`*** y se encarga de ejecutar las ordenes contenidas en el fichero ***`Makefile`***.



## Archivo `Makefile`

- Es el archivo que especifica las reglas y las dependencias para el proceso de compilación.
- Un `Makefile` simple está compuesto por reglas que tienen la siguiente forma:

  <etiqueta>: <lista de dependencias>
  <orden>
  <orden>
  ...
- Nota muy importante: Una línea con una orden empieza siempre con un tabulador (no por varios caracteres espacio).

## `Makefile`: etiquetas, dependencias y órdenes

- Una etiqueta es un rótulo. Generalmente, se refiere a un nombre de fichero. Es el fichero que se quiere actualizar con esta regla.
- También hay etiquetas que indican la acción que la regla realiza. P.ej.: *limpiar*, para indicar que se borrarán todos los archivos objetos.
- Una dependencia es un fichero que se usa en la regla. Si este fichero se modifica la regla se activa.
- Una regla puede depender de varios ficheros.
- Una orden es la acción que `make` realiza si la regla se activa.



## Ejemplo de `Makefile`

- Ejemplo de `makefile` que permite generar un ejecutable llamado "agenda" a partir del código fuente contenido en 3 archivos:

```
todo: agenda
fecha.o: fecha.h fecha.cpp
   g++ -c fecha.cpp -Wall
agenda.o: agenda.cpp fecha.h
   g++ -c agenda.cpp -Wall
agenda: agenda.o fecha.o
   g++ agenda.o fecha.o -o agenda
limpiar:
   rm agenda.o
   rm fecha.o
   rm agenda
```

## Ejemplo: Funcionamiento

- **Uso 1:**

  *prompt$* make

- **Uso 2:**

  *prompt$* make limpiar

- **Uso 3:**

  *prompt$* make fecha.o