**TEMA 15: MÓDULOS.**

Veremos en este tema:

* Introducción a la programación modular, basada en empleo de TAds
* Concepto de módulo, su realización en C+-.
* Características para compilar módulos por separado.
* Desarrollo de programas descomponiéndolos en módulos, es decir, desarrollo modular basado en abstracciones.

1. **Concepto de módulo.**

Un módulo es un fragmento de programa desarrollado de forma independiente, es decir, que pueda ser compilado y probado por separado. Un módulo debe definir un elemento abstracto (o varios relacionados entre sí) y debe ser usado desde fuera con solo saber qué hace el módulo(**interfaz**), pero sin necesidad de conocer como lo hace(**Implementación**).

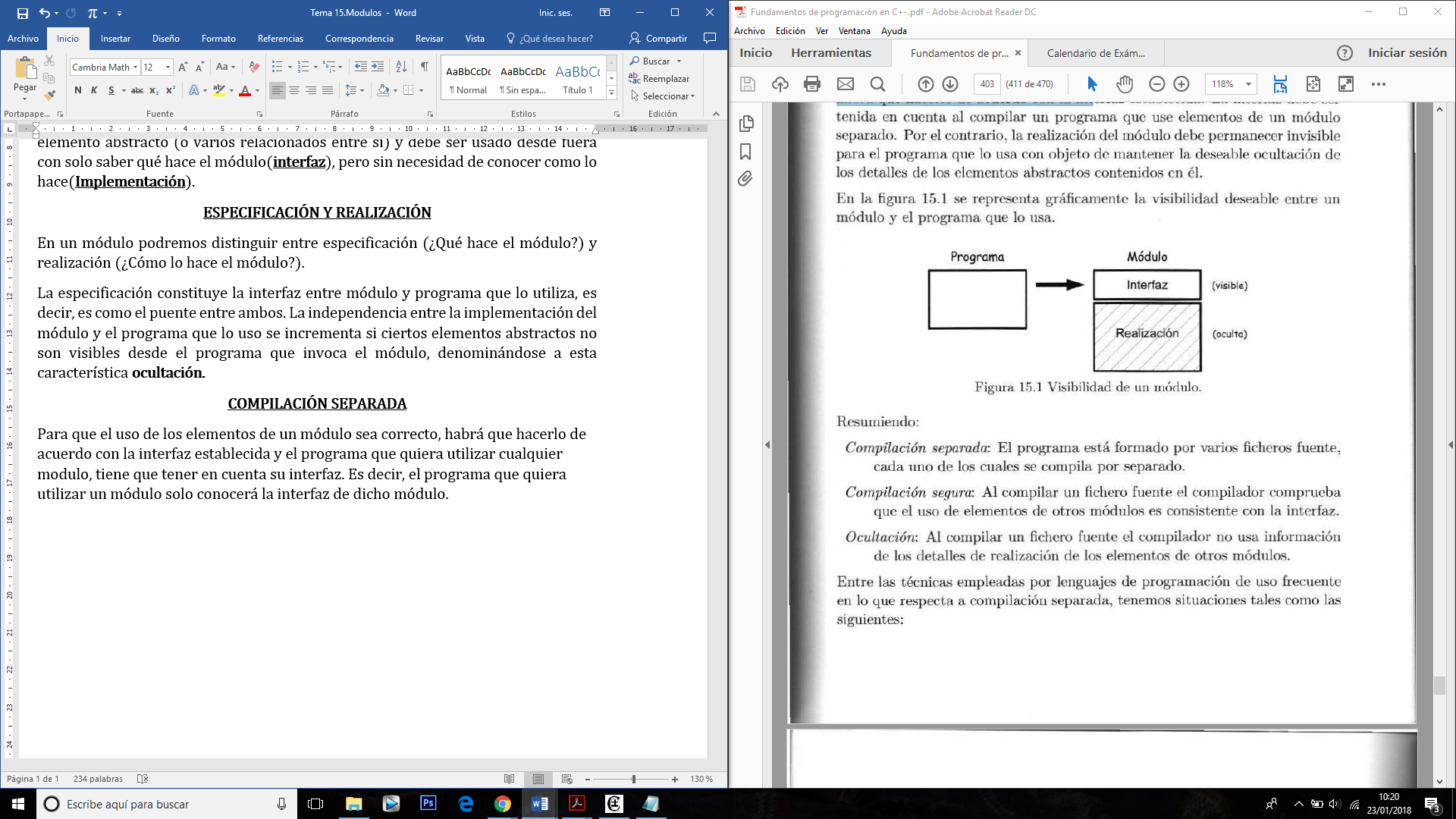
**ESPECIFICACIÓN Y REALIZACIÓN**

En un módulo podremos distinguir entre especificación (¿Qué hace el módulo?) y realización (¿Cómo lo hace el módulo?).

La especificación constituye la interfaz entre módulo y programa que lo utiliza, es decir, es como el puente entre ambos. La independencia entre la implementación del módulo y el programa que lo uso se incrementa si ciertos elementos abstractos no son visibles desde el programa que invoca el módulo, denominándose a esta característica **ocultación.**

**COMPILACIÓN SEPARADA**

Para que el uso de los elementos de un módulo sea correcto, habrá que hacerlo de acuerdo con la interfaz establecida y el programa que quiera utilizar cualquier modulo, tiene que tener en cuenta su interfaz. Es decir, el programa que quiera utilizar un módulo solo conocerá la interfaz de dicho módulo.

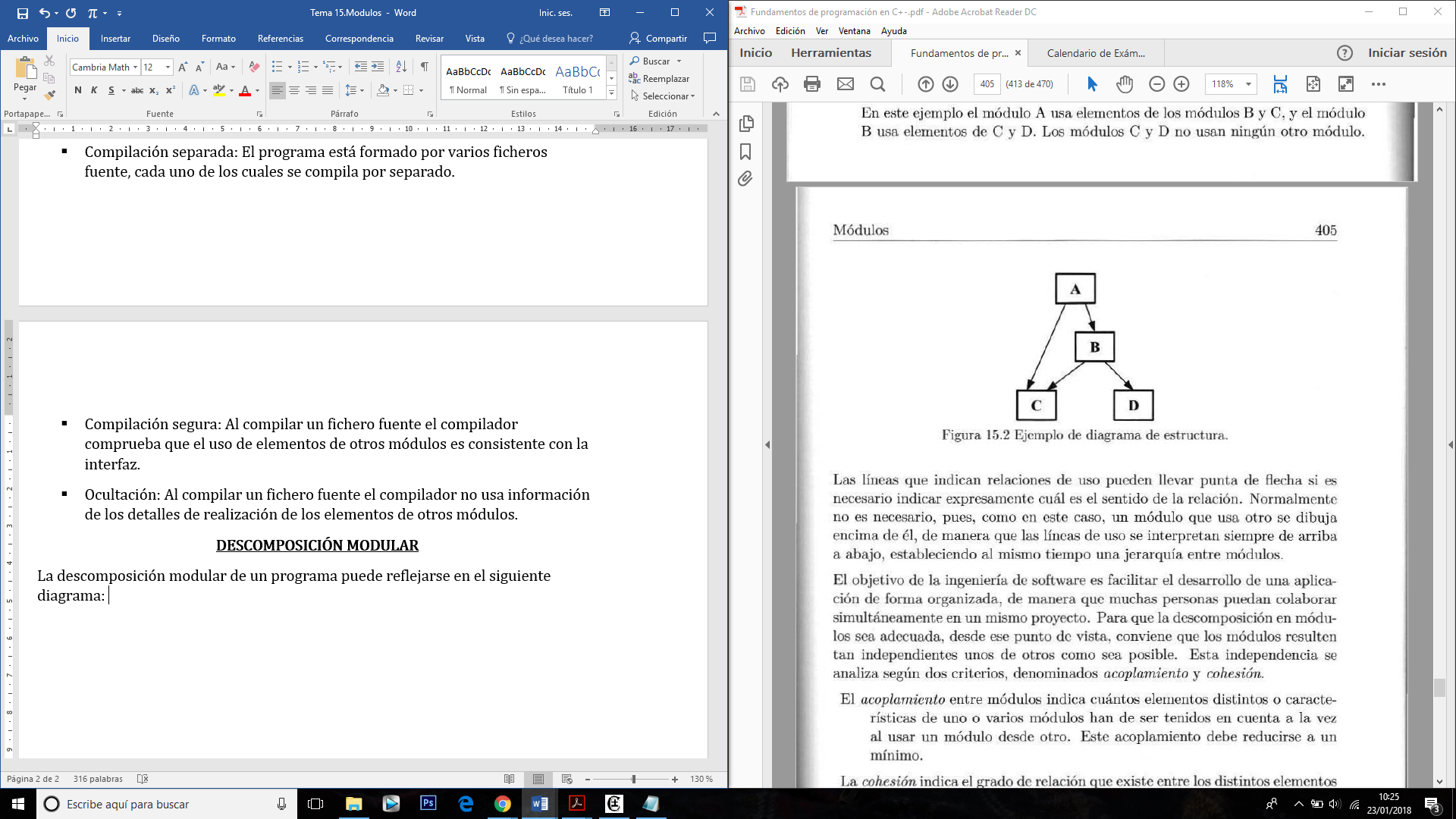


Resumiendo:

* Compilación separada: El programa está formado por varios ficheros fuente, cada uno de los cuales se compila por separado.
* Compilación segura: Al compilar un fichero fuente el compilador comprueba que el uso de elementos de otros módulos es consistente con la interfaz.
* Ocultación: Al compilar un fichero fuente el compilador no usa información de los detalles de realización de los elementos de otros módulos.

**DESCOMPOSICIÓN MODULAR**

La descomposición modular de un programa puede reflejarse en el siguiente diagrama:



En este ejemplo, el módulo A utiliza elementos de B y C, el módulo B utiliza elementos de C y D. Los módulos C y D no utilizan otros elementos. EL objetivo de la ingeniería de software es facilitar el desarrollo de una aplicación de forma organizada, de manera que muchas personas puedan colaborar en un mismo proyecto de manera simultánea. La independencia entre módulos se analiza según dos criterios:

* Acoplamiento: indica cuantos elementos distintos o características de uno o varios módulos se tienen en cuenta a la vez, al usar un módulo desde otro.
* Cohesión: grado de relación que existen entre los distintos elementos de un mismo módulo, y debe ser lo mayor posible. Un mismo módulo no debe incluir elementos que no estén relacionados entre sí.

1. **Módulos en C+-.**

Un programa descompuesto en módulos se escribe un conjunto de ficheros fuentes relacionados entre sí, y pueden compilarse por separado. Cada fichero fuente constituye una unidad de compilación. En c+- la descomposición de un programa en partes se realiza combinando ficheros.

**PROCESO DE COMPILACIÓN SIMPLE**

* Fichero fuente: contiene el código de una unidad de compilación, es decir, es posible invocar el compilador dándole como entrada solo ese fiche fuente(extensión .cpp). La compilación de un fichero fuente, produce un fichero objeto.
* Fichero objeto: Contiene la traducción del código en c+- a instrucciones máquina. Extensión (.o). En general, un fichero objeto no se puede compilar directamente.
* Fichero ejecutable o programa: estos ficheros tienen la extensión (.exe)

**MODULO PRINCIPAL**

Cuando se descompone un programa de c+- en varios módulos, uno de estos módulos ha de ser el programa principal o modulo principal. Este módulo será el que contenga la función **main**(). La ejecución del programa completo consiste en la ejecución del **main**(), pudiendo invocar esta, funciones definidas en otros módulos. El fichero fuente del módulo principal deberá contener el nombre final del programa completo y su extensión es (.cpp).

**MODULOS NO PRINCIPALES**

Los módulos de la aplicación que no contienen una función main() no permiten generar un programa ejecutable por sí solos. Los elementos que contienen están destinados a ser usados en el programa principal o en otros módulos. Hay que distinguir entre elementos públicos(son visibles desde fuera del módulo) y los elementos privados (únicamente son visibles dentro del módulo) . La distinción entre elementos públicos y privados se hace repartiendo el código del módulo en dos ficheros separados.

* Fichero interfaz (extensión .h): Dentro de este fichero hay que poner de cabecera #**pragma once**.
* Fichero Implementación (extensión.cpp): dentro de este fichero hay que insertar la directiva #**include “**Nombre\_archivo.h**”**

NOTA: Ambos ficheros deben contener el mismo nombre, pero con distinta extensión.

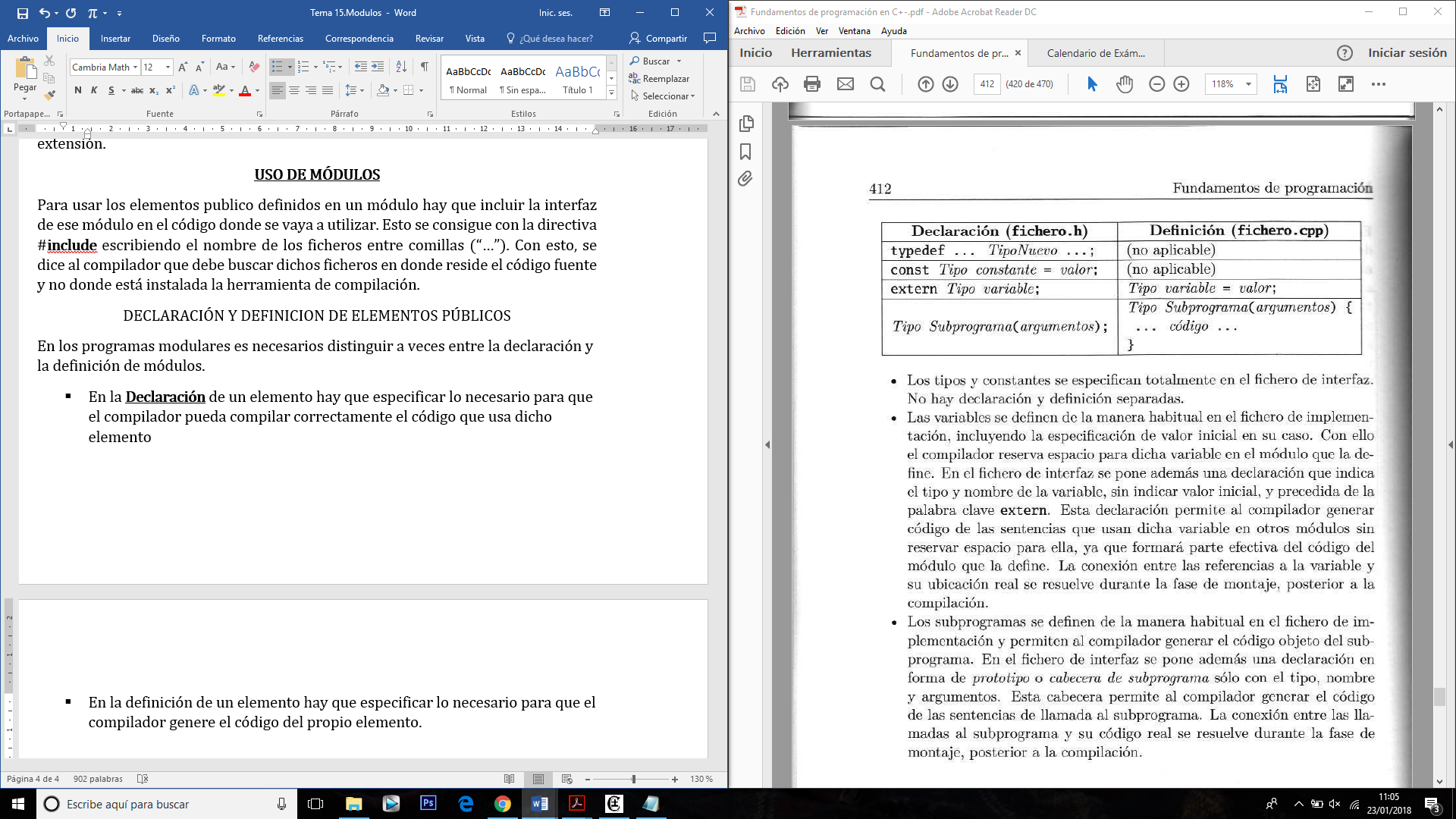
**USO DE MÓDULOS**

Para usar los elementos publico definidos en un módulo hay que incluir la interfaz de ese módulo en el código donde se vaya a utilizar. Esto se consigue con la directiva #**include** escribiendo el nombre de los ficheros entre comillas (“…”). Con esto, se dice al compilador que debe buscar dichos ficheros en donde reside el código fuente y no donde está instalada la herramienta de compilación.

**DECLARACIÓN Y DEFINICION DE ELEMENTOS PÚBLICOS**

En los programas modulares es necesarios distinguir a veces entre la declaración y la definición de módulos.

* En la **Declaración** de un elemento hay que especificar lo necesario para que el compilador pueda compilar correctamente el código que usa dicho elemento
* En la definición de un elemento hay que especificar lo necesario para que el compilador genere el código del propio elemento.



* Los tipos y contantes se especifican totalmente en el fichero de interfaz, no hay declaración y definición separadas.
* La definición e inicialización de variables se definen de manera habitual en el fichero de interfaz. En el fichero de interfaz se pone además un declaración que indica el tipo y nombre de la variable, precedida de la palabra clave **extern.**
* Los subprogramas se definen de la manera habitual en el fichero de implementación y permiten al compilador generar el código objeto del subprograma.

**CONFLICTO DE NOMBRES EN EL ÁMBITO GLOBAL.**

No debe haber nombres repetidos, es decir, no usar el mismo identificar para elementos diferentes de diferentes módulos. Una técnica sencilla para evitar en lo posible los conflictos de nombres públicos globales es asignar a cada módulo un prefijo diferente que se habrá de usar en los nombres de todos sus elementos públicos.

**UNIDADES DE COMPILACION EN C+-**

Los ficheros fuente de los tipos mencionados pueden ser considerados unidades de compilación, en el sentido de que es posible invocar la compilación de cada uno de ellos por separado. Por lo tanto, tendremos com unidades de compilación:

* El modulo principal: main.cpp
* El fichero de interfaz de modulo: modulo.h
* El fichero de implementación de un modulo: modulo.cpp

El significado de la direciva #**include** es similar a copiar y pegar en el punto en el que se invoca la directiva el archivo al que esta se refiere. Por lo tanto, el modulo.h no se compila, ya que tanto el modulo.cpp como el main.cpp utilizan la directiva **include** “interfaz.h” y es como copiarlo y pegarlos en sus archivos.

**COMPILACION DE PROGRAMAS MODULARES. PROYECTOS**

La generación del programa ejecutable final exige:

* Compilar los modulos uno a uno, generando el correpondiente fichero objeto (. o) a partir del fuente (.cpp). Cada compilación individual, usa también los ficheros interfaz (.h) mencionados en la directiva #include en cada módulo (fichero .cpp).
* Se combinan todos los ficheros objeto de los módulos y se monta el programa ejecutable (.exe)

La información mínima que debe contener el fichero de descripción de un proyecto será:

* Nombre del proyecto (= Nombre del programa ejectubale)
* Lista de ficheros fuente de implementación .cpp(incluyendo main)

Opcionalmente:

* Lista de ficheros interfaz .h
* Forma de invocar al compilador.
* Entre otras cosas.

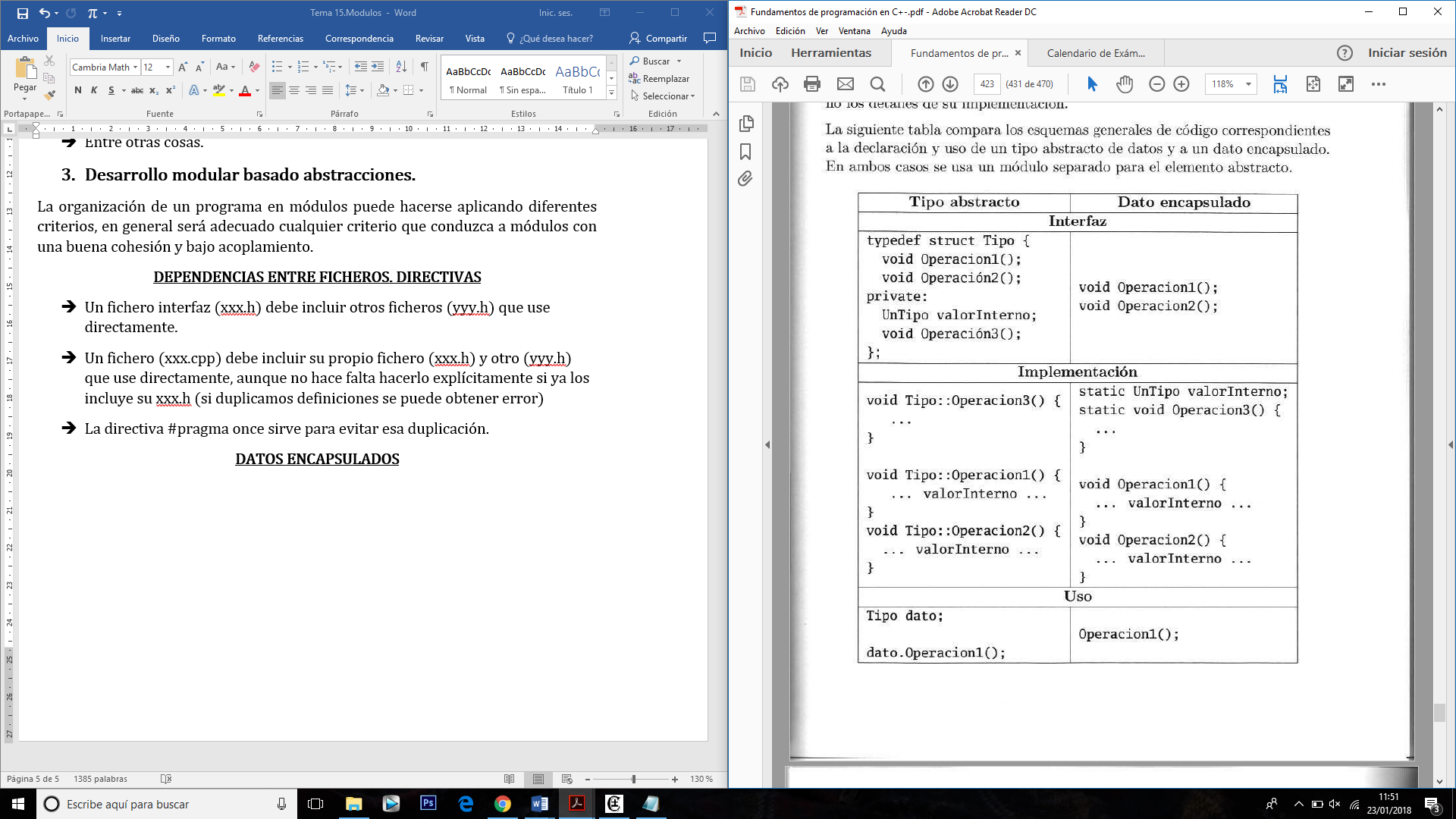
1. **Desarrollo modular basado abstracciones.**

La organización de un programa en módulos puede hacerse aplicando diferentes criterios, en general será adecuado cualquier criterio que conduzca a módulos con una buena cohesión y bajo acoplamiento.

**DEPENDENCIAS ENTRE FICHEROS. DIRECTIVAS**

* Un fichero interfaz (xxx.h) debe incluir otros ficheros (yyy.h) que use directamente.
* Un fichero (xxx.cpp) debe incluir su propio fichero (xxx.h) y otro (yyy.h) que use directamente, aunque no hace falta hacerlo explícitamente si ya los incluye su xxx.h (si duplicamos definiciones se puede obtener error)
* La directiva #pragma once sirve para evitar esa duplicación.

**DATOS ENCAPSULADOS**



**REUTILIZACIÓN DE CODIGO**

Los elementos abstractos de una aplicación pueden ser de utilidad en otras aplicaciones del mismo campo de actividad, especialmente si se procura que dichos elementos abstractos correspondan a elementos reales significativos. Los módulos que definen abstracciones relacionadas entre si pueden agruparse en una biblioteca o librería. Por ejemplo, se pueden combinar módulos que trabajen con fechas, horas, calendarios de diferentes culturas, políticas de cambio de horarios…etc. Todos estos módulos constituirían una biblioteca de utilidad para quienes desarrollen aplicaciones que manejen datos de tiempo oficial.