TEMA 13: PUNTEROS Y VARIABLES DINÁMICAS

Este tema es una introducción a estructuras de datos potencialmente ilimitadas.

1. **Estructura de datos no acotadas.**

Todas las estructuras de datos que hemos definido anteriormente tienen en común que cuando se declara ya se está especificando su tamaño de manera explícita. Por ejemplo:

* En la declaración de un tipo definido **int, char, bool, float** ya sabemos que cada variable que declaremos solo tiene cavidad para un dato.
* En la declaración de vectores el número que esta entre corchetes limita el número de datos que contiene ese vector.
* En el uso de registros(struct) y registros variantes(unión) sabemos que en el registro son el número de campos que contiene y en el registro con variante hay un dato que se selecciona entre los posibles.

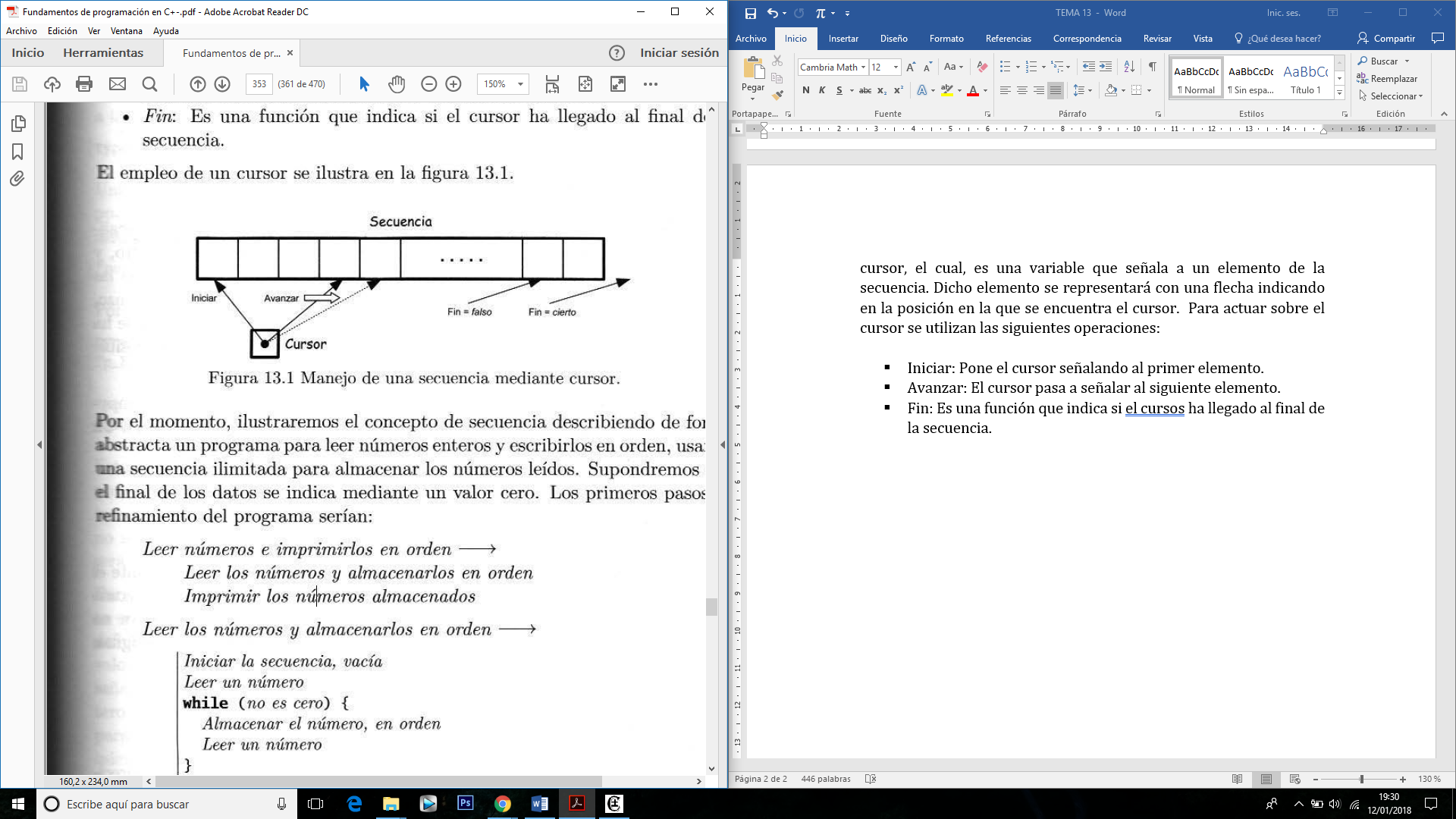
La fijación de un tamaño máximo representa un inconveniente, si el tamaño es relativamente pequeño el programa tendrá una capacidad de tratamiento limitada, si el tamaño se fija a un valor muy grande, el programa será poco eficiente en el uso de la memoria, púes necesitará espacio para toda la estructura de datos, aunque solo se aproveche una pequeña parte.

Tras este análisis, se puede apreciar que sería útil disponer de estructuras de datos que no tuvieran un tamaño fijado de antemano. A este tipo de estructuras de datos se les denomina dinámicas, porque dependiendo de los datos particulares que va recibiendo en la ejecución del programa el tamaño de la estructura de datos va variando, NO ES ESTATICA.

1. **La estructura secuencia.**

La estructura secuencia puede definirse como una estructura de datos de tipo iterativo, pero con un número variable de componentes, es parecida a una formación(vector) con un número variable de elementos. Hay dos tipos de operaciones:

* De construcción:
  + Añadir o retirar componentes al principio de la secuencia.
  + Añadir o retirar componentes al final de la secuencia.
  + Añadir o retirar componentes en posiciones intermedias de la secuencia.
* De acceso:
  + **Acceso Directo**: Se puede acceder a cualquier componente directamente indicando su posición, como en un vector.
  + **Acceso Secuencial:** Los componentes deben tratarse una por una, en el orden en que aparecen en la secuencia. En este caso, se utiliza un cursor, el cual, es una variable que señala a un elemento de la secuencia. Dicho elemento se representará con una flecha indicando en la posición en la que se encuentra el cursor. Para actuar sobre el cursor se utilizan las siguientes operaciones:
    - **Iniciar:** Pone el cursor señalando al primer elemento.
    - **Avanzar:** El cursor pasa a señalar al siguiente elemento.
    - **Fin:** Es una función que indica si el cursor ha llegado al final de la secuencia.



1. **Variables dinámicas.**

Una manera de realizar estructura de datos ilimitadas en C+- es mediante el empleo de variables dinámicas. Las variables dinámicas se crean en el momento necesario, cumplen su función y se destruyen. Las variables dinámicas no tienen nombre, sino que se designan mediante otras variables llamadas punteros.

# DIRECCIONES DE DATOS EN MEMORIA

Para entender mejor los punteros es recomendable, ahondar en el concepto de memoria. El procesador para acceder a cierta posición de memoria envía una serie de pulsos eléctricos(‘0’ = Impulso eléctrico débil o nulo; ‘1’ = Impulso eléctrico fuerte), a cada impulso eléctrico lo denominaremos bit ( 0 ó 1)

En un sistema de 32 bits, envía 32 pulsos eléctricos, por ejemplo, (01001111000111101100110110101001) pera la representación de esta posición de memoria se utiliza el sistema Hexadecimal, en el que cada 8 bits, se representa con los caracteres alfanuméricos (0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F). Por tanto, cada posición de memoria será representada por 8 caracteres Hexadecimales. Un ejemplo de una dirección aleatoria (0x00345ABC).

En los lenguajes de programación, se puede acceder a las direcciones de memoria utilizando **punteros.**

# PUNTEROS

Podemos considerar los punteros como variables que contienen son direcciones de memoria (Valor numérico en sistema hexadecimal) de un tipo de dato concreto, por ejemplo:

**typedef** **int**\* PunteroEnteros; 🡪 Declaración de un Puntero que contiene las direcciones de memoria de datos de tipo entero únicamente.

**int** Variable; 🡪 Declaramos un variable de tipo entero.

PunteroEnteros MyPuntero; 🡪 Inicializamos el puntero en nuestro programa o sub-programa.

PunteroEnteros MyPuntero2; 🡪 Se inicializa un puntero con una posición de memoria determinada por el compilador.

MyPuntero = &Variable; 🡪 Asignamos al puntero la dirección en la que se encuentra la variable denominada “Variable”.

\*MyPuntero = 30; 🡪 Le damos un valor concreto a la variable que está en la dirección que en ese momento tiene el puntero.

\*MyPuntero2 = 56 🡪 Asignamos valor numérico a la posición que indica (MyPuntero2)

**&**PUNTERO = POSICIÓN EN SISTEMA HEXADECIMAL DEL PUNTERO.

**\***PUNTERO = VALOR NÚMERICO, CARÁCTER …ETC DE LA POSICIÓN EN MEMORIA QUE TIENE EL PUNTERO EN ESE MOMENTO.

En caso de que el puntero no apunte a una variable dinámica, por ejemplo, se crea un puntero y no se le asigna la posición de una variable concreta, el resultado será imprevisible. Para comprobar si el puntero está apuntando a una variable dinámica, se usa el valor especial NULL (está definido en la librería **stdlib.h** y en otras librerías). Este valor indica que el puntero no señala a la posición de una variable dinámica.

NOTA: *Cuando Inicializamos un puntero, es recomendable asignarle el valor NULL. Así podremos controlar si el puntero tiene asignado una variable o no.*

# USO DE VARIABLES DINÁMICAS

Características de variables estáticas o limitadas en tamaño:

* Tienen una dirección de memoria reservada de antemano al comienzo de la ejecución del programa (Por ello, la zona de declaración de datos es antes de la zona de ejecución).
* Las variables creadas localmente en un sub-programa desaparecen al salir del sub-programa.
* Tiene un nombre asignado para localizarla.

Características de variable dinámica:

* No tienen espacio de memoria reservado.
* Las posiciones de memoria donde se guardan los datos de las variables dinámicas se crean en el momento que se indique (DURANTE LA EJECUCIÓN DEL PROGRAMA).
* Si se crea una variable dinámica en un sub-programa, al terminar el sub-programa continúa existiendo. (A no ser que se especifique lo contrario).
* No tiene ningún nombre asignado, la única manera de acceder a ellas es mediante punteros.

Para crear una variable dinámica utilizamos el operador **new:**

**typedef** Tipo-de-variable\* Tipo-puntero; 🡪 Creamos puntero para un determinado tipo de variable.

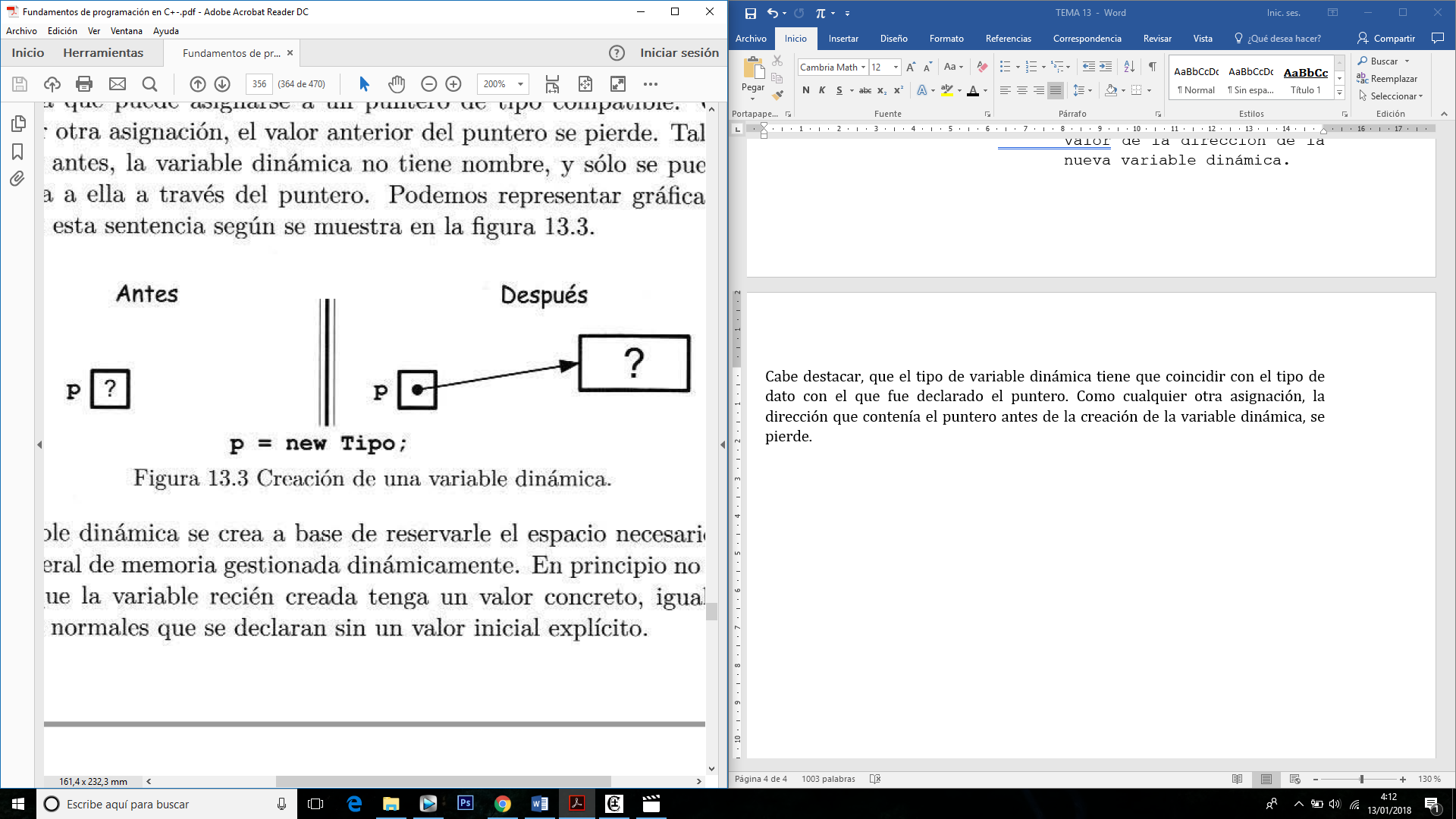
Tipo-puntero **puntero**; 🡪 Inicializamos puntero(el compilador asignara dirección de memoria).

**puntero** = NULL; 🡪 Nosotros eliminamos la dirección del puntero, o asignamos a esa dirección el valor = 0x00000000.

**puntero** = **new** Tipo-de-variable; 🡪 Se asigna a “puntero” el valor de la dirección de la nueva variable dinámica.

\***puntero** = 33**;** 🡪Se le asigna a la variable dinámica que creamos en la anterior sentencia el valor 33.

Cabe destacar, que el tipo de variable dinámica tiene que coincidir con el tipo de dato con el que fue declarado el puntero. Como cualquier otra asignación, la dirección que contenía el puntero antes de la creación de la variable dinámica, se pierde.



Una vez creada la variable dinámica, se le ha guardado un especio en memoria, pero no se le ha asignado ningún valor directamente. Las variables dinámicas existirán durante todo el programa, a no ser que se indique explícitamente que ya no son necesarias con la sentencia **delete** que permite destruir la variable dinámica a la que señala el puntero.

**delete** puntero; 🡪 Eliminamos la variable dinámica apuntada pero no se elimina la dirección a la que señala el puntero.

puntero = NULL; 🡪 En este caso, estamos eliminando la dirección que guarda el puntero.

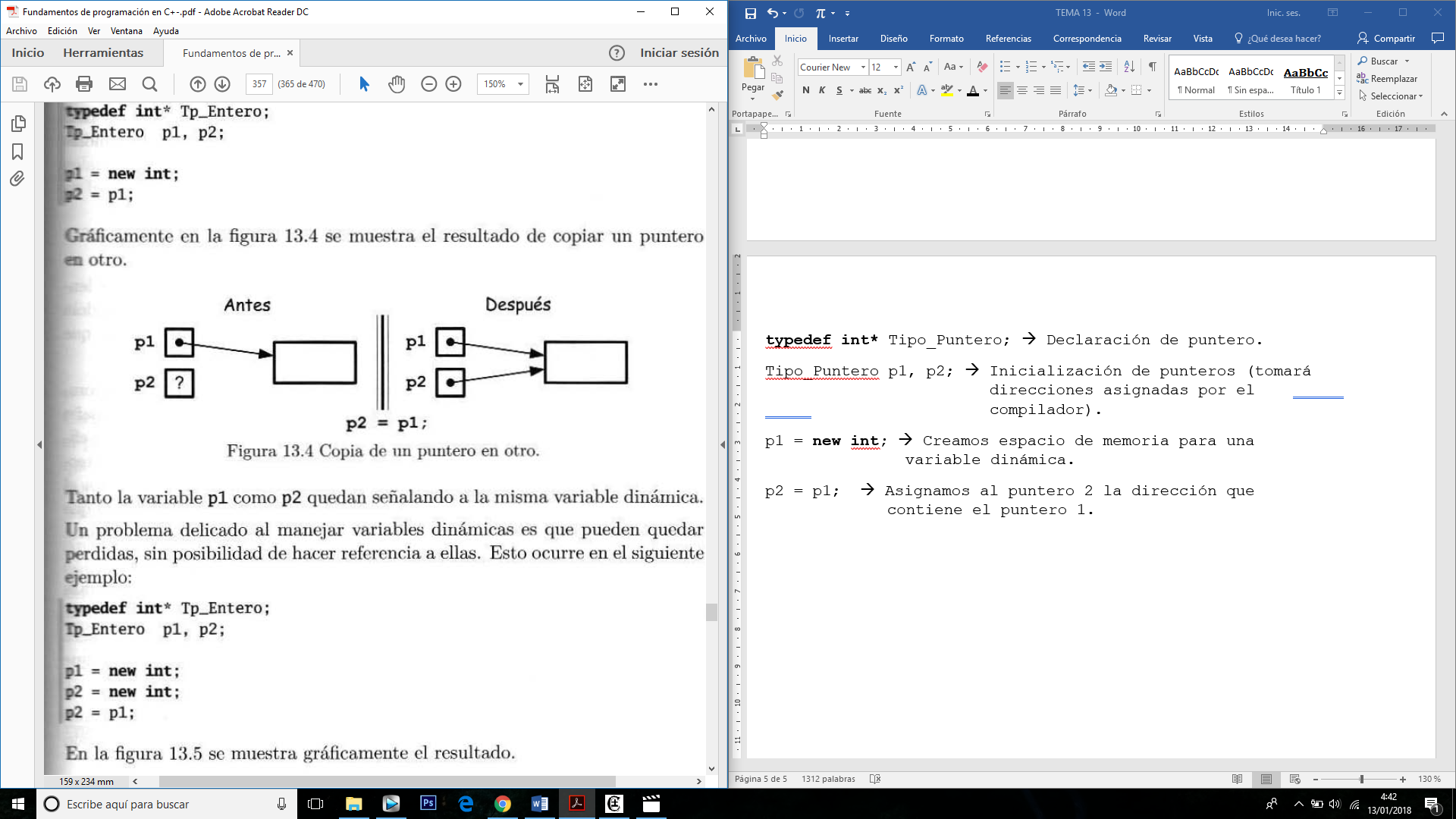
Podemos designar la misma posición a varios punteros, haciendo una igualación, por ejemplo:

**typedef int\*** Tipo\_Puntero; 🡪 Declaración de puntero.

Tipo\_Puntero p1, p2; 🡪 Inicialización de punteros (tomará direcciones asignadas por el compilador).

p1 = **new int;** 🡪 Creamos espacio de memoria para una variable dinámica.

p2 = p1; 🡪 Asignamos al puntero 2 la dirección que contiene el puntero 1.



Tanto el puntero p1 como el p2, apuntan a la misma variable dinámica. Un problema al manejar variables dinámicas es que pueden quedar perdidas, sin posibilidad de recuperarlas (ya que no tienen un nombre asignado como referencia para encontrarlas, si no una dirección de 8 caracteres hexadecimales difíciles de memorizar para el programador). Por ejemplo:

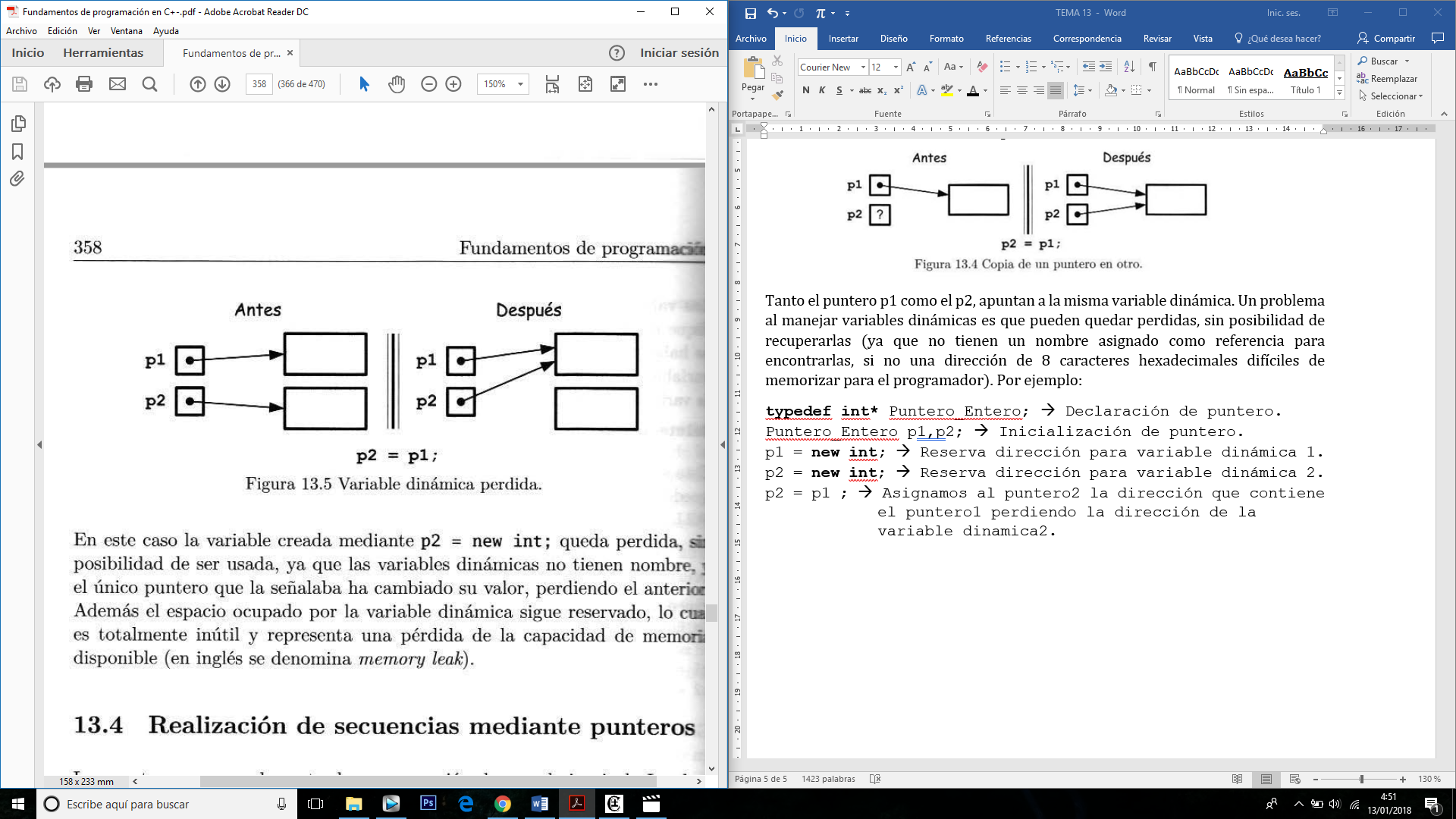
**typedef int\*** Puntero\_Entero; 🡪 Declaración de puntero.

Puntero\_Entero p1,p2; 🡪 Inicialización de puntero.

p1 = **new int;** 🡪 Reserva dirección para variable dinámica 1.

p2 = **new int**; 🡪 Reserva dirección para variable dinámica 2.

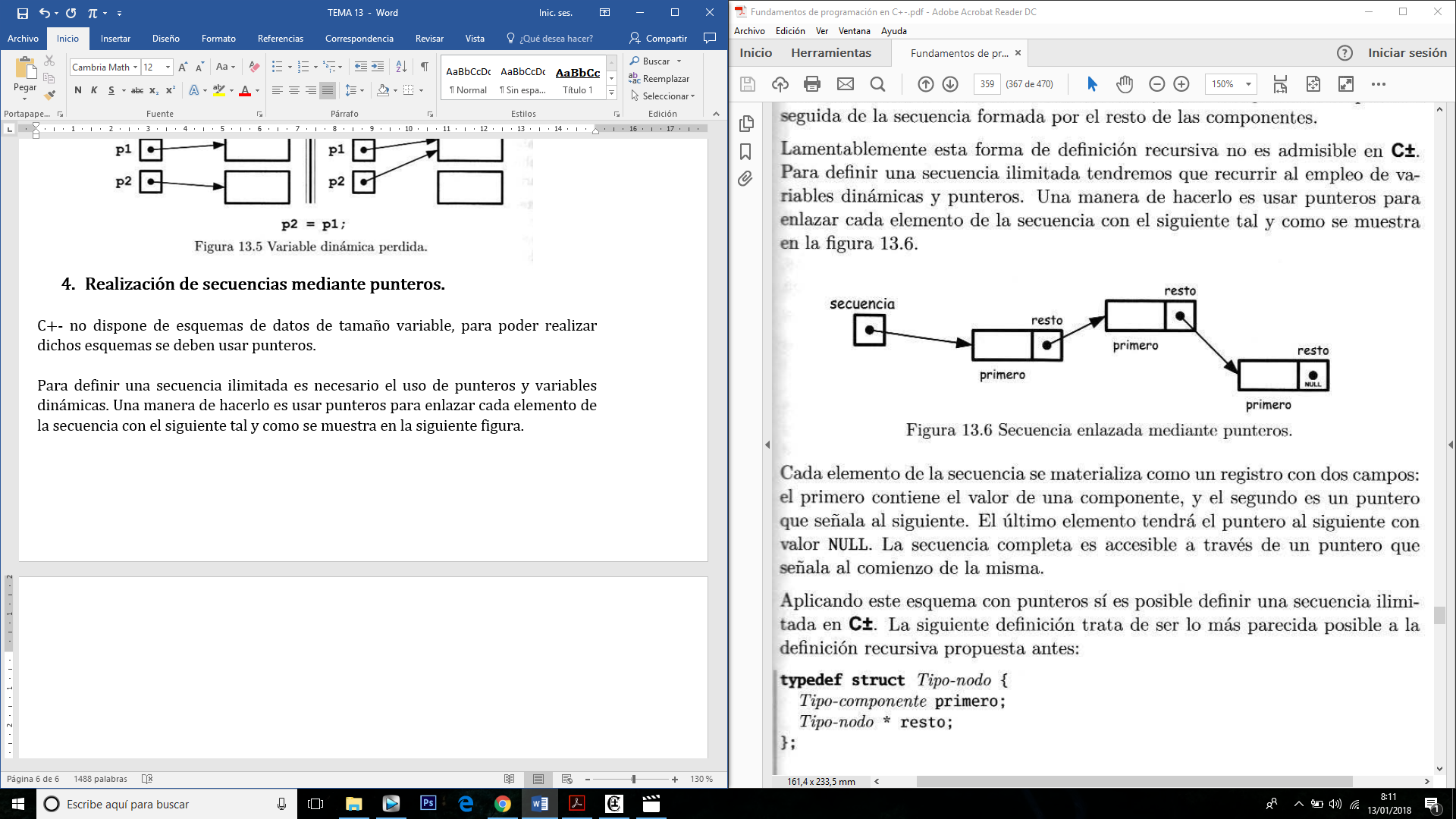
p2 = p1 ; 🡪 Asignamos al puntero2 la dirección que contiene el puntero1 perdiendo la dirección de la variable dinamica2.



1. **Realización de secuencias mediante punteros.**

C+- no dispone de esquemas de datos de tamaño variable, para poder realizar dichos esquemas se deben usar punteros.

Para definir una secuencia ilimitada es necesario el uso de punteros y variables dinámicas. Una manera de hacerlo es usar punteros para enlazar cada elemento de la secuencia con el siguiente tal y como se muestra en la siguiente figura.



Para entenderlo mejor, tendremos que crear un registro (**struct**) que contenga dos datos, una variable que indique el valor de un dato en una determinada dirección + un puntero que indique la posición del siguiente **struct**, el cual, a su vez contiene otro dato y la dirección del siguiente **struct**. Si se quiere indicar que después del **struct** tratado no hay otro elemento de la secuencia, se introduce en su puntero el valor NULL.

**typedef struct** Tipo-nodo {

tipo-componente primero;🡪 Valor del dato en ese nodo.

tipo-nodo\* resto; 🡪 Puntero que señala a la dirección del siguiente nodo.

};

**typedef** Tipo-nodo\* Tipo-secuencia;

**TIPO SECUENCIA**

|  |  |  |
| --- | --- | --- |
| DIRECCIÓN EN MEMORIA | DATO ALMACENADO | PUNTERO DE ESA POSICIÓN |
| 0x0000002F | 1234 | NULL |
| 0x00000028 | 56 | 0x0000002F |
| 0x0000001F | 23 | 0x00000028 |
| 0x00000018 | 1 | 0x0000001F |
| 0x0000000F | 32 | 0x00000018 |
| 0x00000008 | 25 | 0x0000000F |
| 0x00000000 | 12 | 0x00000008 |

NOTA: Estas direcciones de memoria son un ejemplo, el compilador se encarga de elegir las direcciones.

Para nombrar campos dentro del nodo hay dos notaciones disponibles.

**typedef struct** Tipo-nodo {

tipo-componente primero;🡪 Valor del dato en ese nodo.

tipo-nodo\* resto; 🡪 Puntero que señala a la dirección del siguiente nodo.

};

**typedef** Tipo-nodo\* Tipo-secuencia;

Tipo-componente valor;

Tipo-secuencia secuencia, siguiente;

MANERA 1:

**if**(secuencia != NULL){

(\*secuencia). primero = valor; 🡪 Asignamos valor al elemento del nodo “primero”.

siguiente = (\*secuencia).resto;

}

MANERA 2:

**if**(secuencia != NULL){

secuencia->primero = valor; 🡪 Asignamos valor al elemento del nodo “primero”.

siguiente = secuencia->resto;

# OPERACIONES CON SECUENCIAS ENLAZADAS

Suponemos la existencia de un cursor que va señalando a las componentes una tras otra de una secuencia enlazada. El cursor será un puntero.

* **DEFINICIÓN:** La definición de la secuencia es como la declaración previamente observada.

**typedef** **struct** TipoNodo {

**int** valor;

**TipoNodo**\* siguiente; Declaración.

};

**typedef** TipoNodo \*TipoSecuencia;

TipoSecuencia secuencia; 🡪 Inicialización

* **RECORRIDO:**

Para recorrer una secuencia es necesario mediante un bucle de acceso a elementos y avance del cursor. Como una secuencia tiene un número indefinido de elementos no se usará un bucle de contador. Usaremos un **while.** Como ejemplo:

**typedef** **struct** TipoNodo {

**int** valor;

TipoNodo\* siguiente;

};

**typedef** TipoNod\* TipoSecuencia;

**typedef** TipoNodoTipoPuntNodo;

TipoSecuencia MySecuencia;

TipoPuntNodo cursor;

cursor = secuencia;🡪 Copio dirección de cursor en puntero secuencia

**while (**cursor != NULL**){**

printf**(**“%5d”, curso->valor**);**

cursor = cursor ->siguiente;

**}**

* **BÚSQUEDA:** la búsqueda ha de realizarse de forma secuencial, la búsqueda es igual al recorrido (con un búcle **while**) pero con dos condiciones, una que limite el recorrido hasta el final de la secuencia, y otra que analiza la condición que queramos que se cumple para identificar el elemento buscado.

**int** numero; */\*Valor a buscar\*/*

TipoPuntNodo cursor, anterior;

cursor = secuencia;

anterior == NULL;

while (cursor != NULL && cursor->valor < numero) {

anterior == cursor;

cursor == cursor->siguiente;

}

* **INSERCIÓN:** La inserción de un nuevo elemento se consigue creando una variable dinámica para contenerlo, y modificando los punteros para enlazar dicha variable dentro de la secuencia. El caso más sencillo es insertar un elemento detrás de uno nuevo dado, Por ejemplo:

**int** numero;

TipoPuntNodo cursor, anterior, nuevo;

nuevo = **new** TipoNodo; 🡪 1º Paso: Crear Nuevo nodo.

nuevo->valor = numero; 🡪 Asignamos valor al nodo.

nuevo->siguiente = anterior->siguiente;

anterior->siguiente = nuevo;

El primer paso a seguir es crear el nuevo nodo, el compilador designara una posición que nosotros no conocemos de antemano a este nuevo nodo. Nuestro trabajo consiste en modificar la dirección del nuevo nodo y que se adapte a la secuencia que hemos creado, para ello, asignamos al puntero de este nuevo nodo la posición del anterior nodo,

NUESTRA SECUENCIA ANTES DE CUALQUIER MODIFICACIÓN

NODO 1

VALOR DATO

DIRECCIÓN NODO 2

NODO 2

VALOR DATO

DIRECCIÓN NODO 3

NODO 3

VALOR DATO

DIRECCIÓN NODO 4

NUESTRA SECUENCIA AL CREAR UN NUEVO NODO

**DEBEMOS ROMPER ESTE ENLACE**

NODO 1

VALOR DATO

DIRECCIÓN NODO 2

NODO 2

VALOR DATO

DIRECCIÓN NODO 3

NODO 3

VALOR DATO

DIRECCIÓN NODO 4

* **PASO 1: CREAR NUEVO NODO**
* **PASO 2: ASIGNAR A NUEVO NODO DIRECCIÓN QUE CONTENIA ANTERIOR NODO**

## NODO NUEVO

VALOR DATO

DIRECCIÓN ALEATORIA

NODO 1

VALOR DATO

DIRECCIÓN NODO 2

NODO 2

VALOR DATO

DIRECCIÓN NODO 3

## NODO NUEVO

VALOR DATO

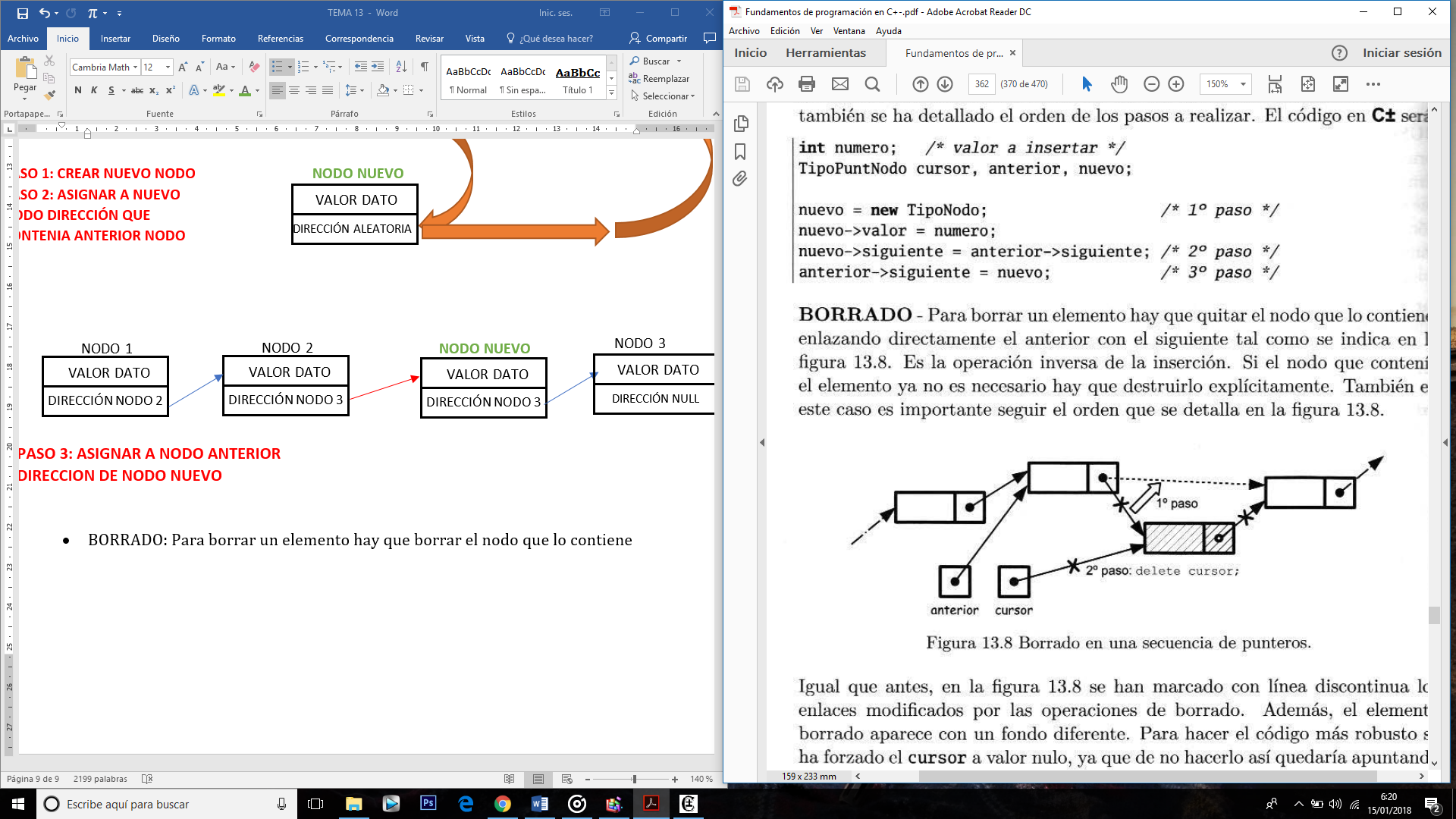
DIRECCIÓN NODO 3

NODO 3

VALOR DATO

DIRECCIÓN NULL

* PASO 3: ASIGNAR A NODO ANTERIOR DIRECCION DE NODO NUEVO
* **BORRADO**: Para borrar un elemento hay que borrar el nodo que lo contiene.



TipoPuntNodo cursor, anterior; 🡪 Creamos un puntero que este situado en el nodo a eliminar, y otro puntero que señale al anterior nodo del que se va a eliminar.

anterior->siguiente = cursor ->siguiente;

delete cursor;

cursor = NULL; 🡪 SE ELIMINA POSICIÓN GUARDADA EN PUNTERO CURSOR;

1. Punteros y paso de argumentos.

Los punteros se pueden pasar como argumento a otros sub-programas. Normalmente cuando pasamos punteros a un sub-programa estos no se podrán modificar a no ser que lo especifiquemos

int Mifuncion (TipoPuntero Mysecuencia, TipoPuntero &Mypuntero){

sentencias;

}

Mysecuencia y Mypuntero son del mismo tipo de dato definido, pero Mypuntero lleva un ‘&’ delante, por lo que las modificaciones que reciba Mypuntero dentro del subprograma se verán repercutidas en el programa principal.