**PONTIFICIA UNIVERSIDAD JAVERIANA DE CALI**

**BRENDA DAYANA TORRES MURCIA**

**SEGUNDO SEMESTRE**

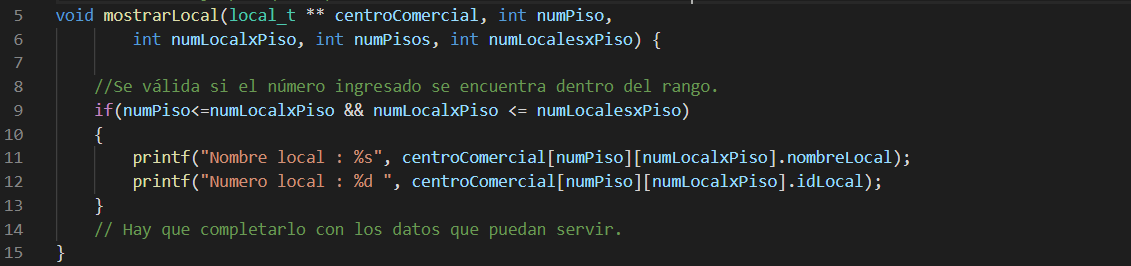
**INGENIERÍA EN SISTEMAS Y COMPUTACIÓN**

**TÉCNICAS Y PRÁCTICAS DE PROGRAMACIÓN**

**LUISA FERNANDA**

**2020**

**EXPLICACIÓN:**

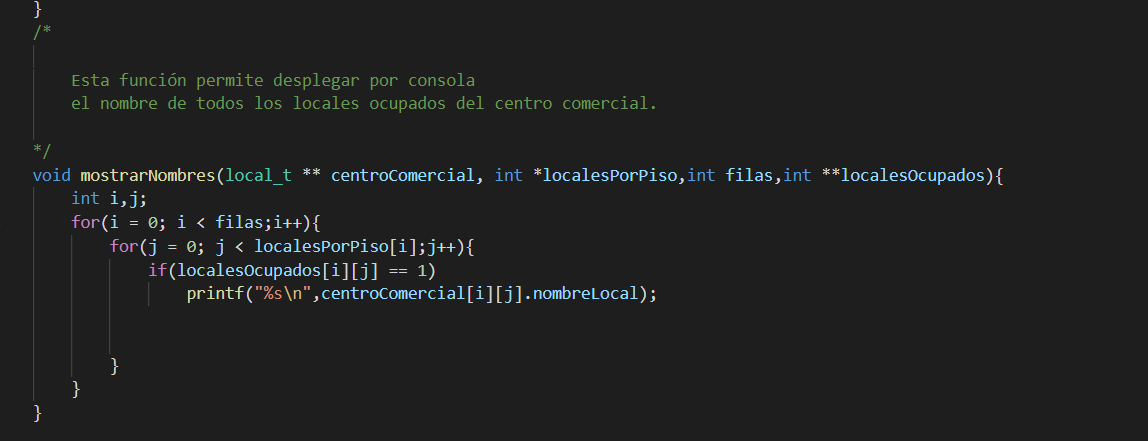


**FUNCIÓN MOSTRARLOCAL:**

**Entradas:** Se recibe el número de local por piso, número de pisos y números de locales por pisos para poder acomodarlos para luego mostrarse.

**Salidad:** Muestra el nombre y el número del local en el que se encuentra situado.

**Conceptos usados:** structs, matrices, ciclos, funciones, “punteros” y condicionales.

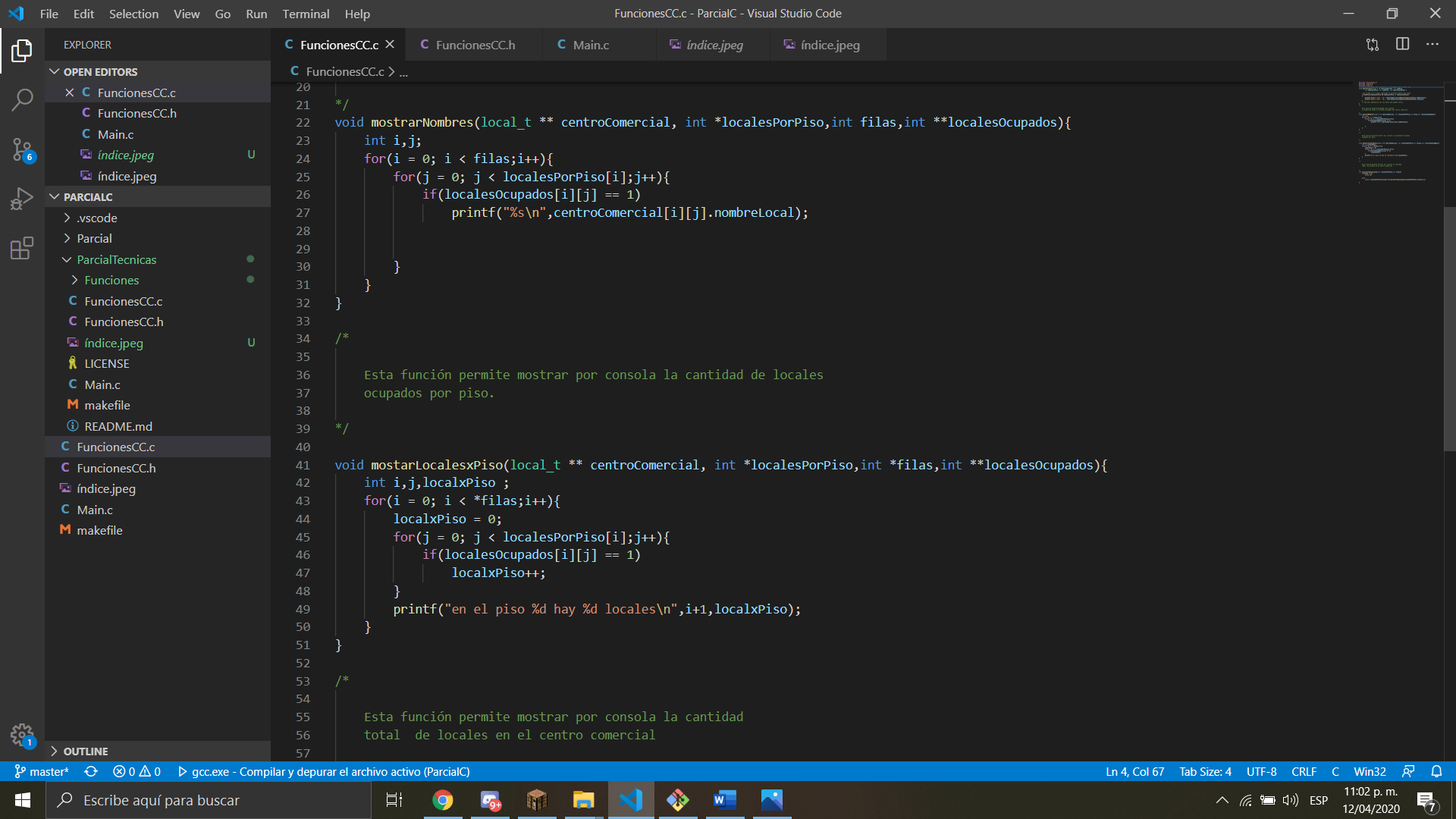


**FUNCIÓN MOSTRARNOMBRES:**

**Entradas:** Se recibe los datos de la struct [Matriz], los locales por piso, las filas y los locales ocupados, lo que realiza es un ciclo para recorrerlo y saber que locales hay dentro de la matriz del centro comercial.

**Salidad:** Muestra la posición del local y el nombre del local.

**Conceptos usados:** structs, matrices, ciclos, funciones, “punteros” y condicionales.

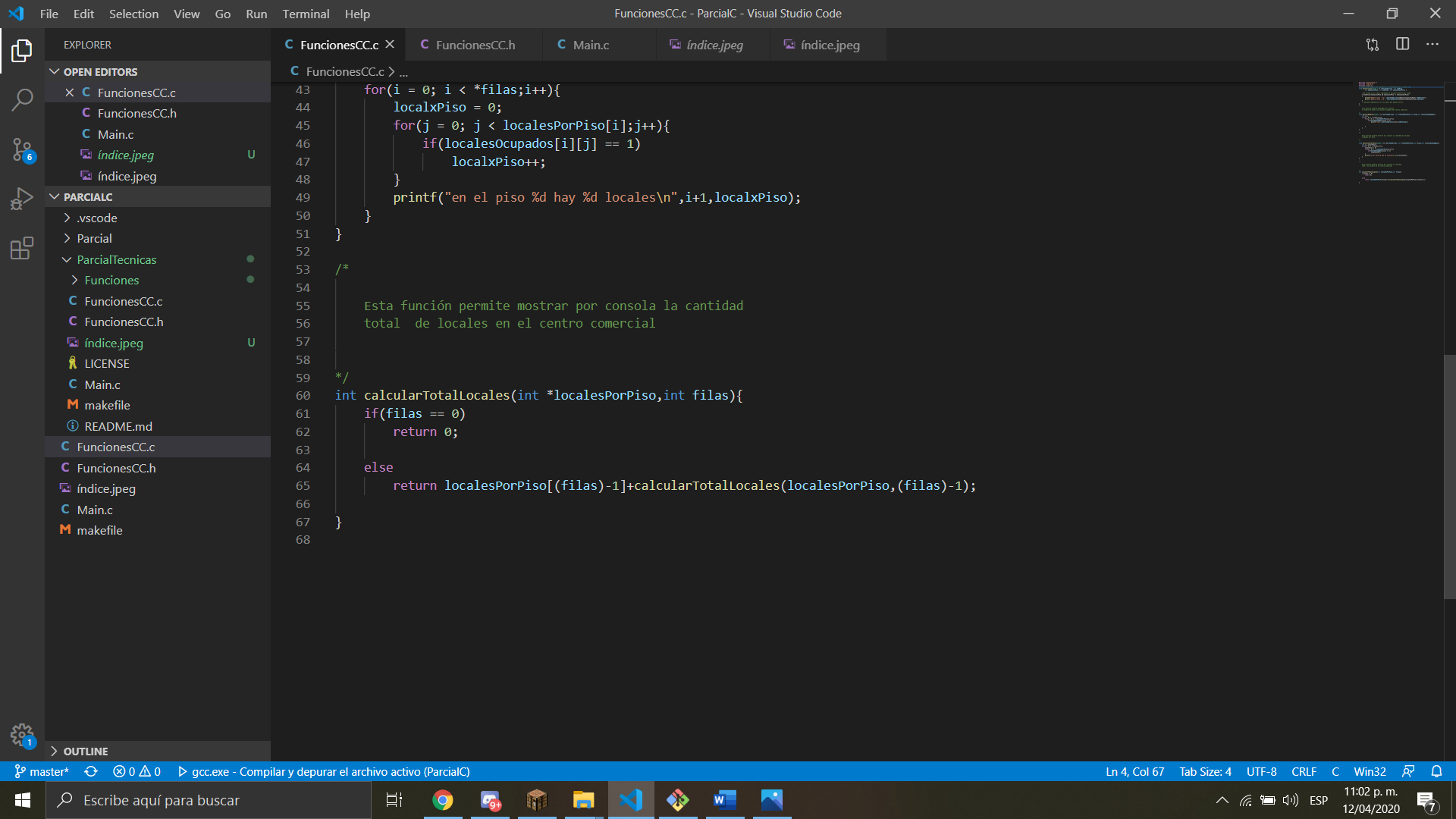


**FUNCIÓN MOSTRARLOCALESXPISO:**

**Entradas:** Se recibe los datos de la struct [Matriz], los locales por piso, las filas y los locales ocupados, lo que realiza es un ciclo para recorrerlo y saber que locales hay.

**Salidad:** Muestra el piso y el local que hay en estos y se le suma uno para

**Conceptos usados**: structs, matrices, ciclos, funciones, “punteros”.



**FUNCIÓN CALCULARTOTALLOCALES:**

**Entradas:** Se recibe los locales por piso y las filas, si hay cero retornará que no hay pero en caso de haber con locales por piso que se le pide al usuario, se le irá restando hasta no tener más.

**Salidad:** Arroja la cantidad total de locales.

**Conceptos usados:** structs, matrices, condicionales, funciones.

enum Bool {False,True};  // Lo uso en el arreglo locales ocupados para marcar como True o False.

Coloqué un enum bool, que sería un booleano de verdadero y falso, para luego utilizarlo en locales ocupados, que marqué si está o no ocupado.

case 1:

                printf("Ingrese la cantidad de pisos del centro comercial\n");

                localesPorPiso = malloc(filas\*sizeof(int));

                centroComercial = malloc(filas\* sizeof(local\_t \*));

                localesOcupados = malloc(filas\*sizeof(int \*));

                scanf("%d",&filas);

                for(i = 0; i < filas;i++){

                    printf("Ingrese los locales disponibles en el piso %d\n",i+1);

                    scanf("%d",&columna);

                    localesPorPiso[i] = columna;

                    centroComercial[i] = malloc(columna\*sizeof(local\_t));

                    localesOcupados[i] = malloc(columna\*sizeof(int));

                    for(j = 0; j < columna;j++)

                        localesOcupados[i][j] = False;

                }

                break;

**Caso 1:** Asignar la cantidad de pisos y locales dentro del centro comercial.

**Entradas:** Se pide al usuario que ingrese la cantidad de pisos del centro comercial para luego asignarles la memoria con el malloc y junto al sizeof devolver el tamaño, acomoda dentro de la struct este valor que introdujo, luego se le pide que ingrese la cantidad de locales disponibles en el piso que anteriormente agregó y se le agrega a las columnas, dependiendo de esas columnas lo que hace es acomodarlo en la matriz de structs que se realizó anteriormente.

**Salidas:** Cantidad de pisos del centro comercial y la cantidad de los locales disponibles en esos pisos.

**Temas Aplicados:** Switch, ciclos, matrices, funciones y structs.

case 2:

                printf("Ingrese el piso donde quiere colocar el local\n");

                scanf("%d",&fila);

                while(fila < 0 || fila >= filas){

                    printf("Ingrese una fila entre 0 y %d\n",filas);

                    scanf("%d",&fila);

                }

                printf("Ingrese el local donde quiere poner el local comercial\n");

                scanf("%d",&columna);

                while(columna < 0 || columna > localesPorPiso[fila]){

                    printf("Ingrese una columna entre 0 y %d\n",localesPorPiso[fila]);

                    scanf("%d",&columna);

                }

                while(localesOcupados[fila][columna] == 1){

                    printf("Ingrese otro local ya que este esta ocupado\n");

                    printf("Ingrese el piso donde quiere colocar el local\n");

                    scanf("%d",&fila);

                    while(fila < 0 || fila >= filas){

                        printf("Ingrese una fila entre 0 y %d\n",filas);

                        scanf("%d",&fila);

                    }

                    printf("Ingrese el local donde quiere poner el local comercial\n");

                    scanf("%d",&columna);

                    while(columna < 0 || columna > localesPorPiso[fila]){

                        printf("Ingrese una columna entre 0 y %d\n",localesPorPiso[fila]);

                        scanf("%d",&columna);

                    }

                }

                localesOcupados[fila][columna] = True;

                printf("Ingrese el nombre del local\n");

                scanf("%s",centroComercial[fila][columna].nombreLocal);

                printf("%s\n",centroComercial[fila][columna].nombreLocal);

                centroComercial[fila][columna].idLocal = ids++;

                centroComercial[fila][columna].pisoLocal = fila;

                centroComercial[fila][columna].numLocalxPiso = columna;

                break;

**Caso 2:** Ingresar un nuevo local al centro comercial.

**Entradas:** Se pide al usuario que ingrese la cantidad de pisos del centro comercial para luego asignarles la memoria con el malloc y junto al sizeof devolver el tamaño, acomoda dentro de la struct este valor que introdujo, luego se le pide que ingrese la cantidad de locales disponibles en el piso que anteriormente agregó y se le agrega a las columnas, dependiendo de esas columnas lo que hace es acomodarlo en la matriz de structs que se realizó anteriormente, lo que se realiza es ir acomodando cada valor de los que ha ingresado en la matriz ya hecha.

**Salidas:** Nuevo local con su ubicación.

**Conceptos usados:** structs, matrices, ciclos, funciones, “punteros”.

  case 3:

                mostrarNombres(centroComercial, localesPorPiso, filas,localesOcupados);

                break;

**Caso tres se utiliza la función que anteriormente expliqué.**

case 4:

                mostarLocalesxPiso(centroComercial, localesPorPiso, &filas,localesOcupados);

                break;

**Caso cuatro se utiliza la función que anteriormente expliqué.**

case 5:

                printf("Ingrese el piso del local que quiere cambiar el nombre\n");

                scanf("%d",&fila);

                while(fila < 0 || fila >= filas){

                    printf("Ingrese una fila entre 0 y %d\n",filas);

                    scanf("%d",&fila);

                }

                printf("Ingrese la columna del local que quiere cambiar el nombre\n");

                scanf("%d",&columna);

                while(columna < 0 || columna > localesPorPiso[fila]){

                    printf("Ingrese una columna entre 0 y %d\n",localesPorPiso[fila]);

                    scanf("%d",&columna);

                }

                while(localesOcupados[fila][columna] == 0){

                    printf("Ingrese otro local que este ocupado\n");

                    printf("Ingrese el piso del local que quiere cambiar el nombre\n");

                    scanf("%d",&fila);

                    while(fila < 0 || fila >= filas){

                        printf("Ingrese una fila entre 0 y %d\n",filas);

                        scanf("%d",&fila);

                    }

                    printf("Ingrese la columna del local que quiere cambiar el nombre\n");

                    scanf("%d",&columna);

                    while(columna < 0 || columna > localesPorPiso[fila]){

                        printf("Ingrese una columna entre 0 y %d\n",localesPorPiso[fila]);

                        scanf("%d",&columna);

                    }

                }

                printf("nombre viejo %s\n",centroComercial[fila][columna].nombreLocal);

                printf("Ingrese nuevo nombre\n");

                scanf("%s",centroComercial[fila][columna].nombreLocal);

                break;

**Entradas:** Recibe el piso y la columna del local que se va a cambiar y con estos datos verificar si existe tal local para luego mostrar el nombre viejo y pedir el nombre nuevo que agregará.

**Salidas:** Nombre nuevo del local.

**Conceptos usados:** switch, structs, matrices, ciclos y funciones.

case 6:

                printf("Ingrese el piso del local que quiere eliminar\n");

                scanf("%d",&fila);

                while(fila < 0 || fila >= filas){

                    printf("Ingrese una fila entre 0 y %d\n",filas);

                    scanf("%d",&fila);

                    }

                printf("Ingrese la columna del local que quiere eliminar\n");

                scanf("%d",&columna);

                while(columna < 0 || columna > localesPorPiso[fila]){

                    printf("Ingrese una columna entre 0 y %d\n",localesPorPiso[fila]);

                    scanf("%d",&columna);

                    }

                localesOcupados[fila][columna] = False;

**Entradas:** Recibe el piso y columna del local a eliminar y va a verificar que estos existan para luego eliminar.

**Salidas:** Locales.

**Conceptos usados:** switch, structs, matrices, ciclos y funciones.

  case 7:

                printf("Cantidad de locales totales %d\n",calcularTotalLocales(localesPorPiso,filas) );

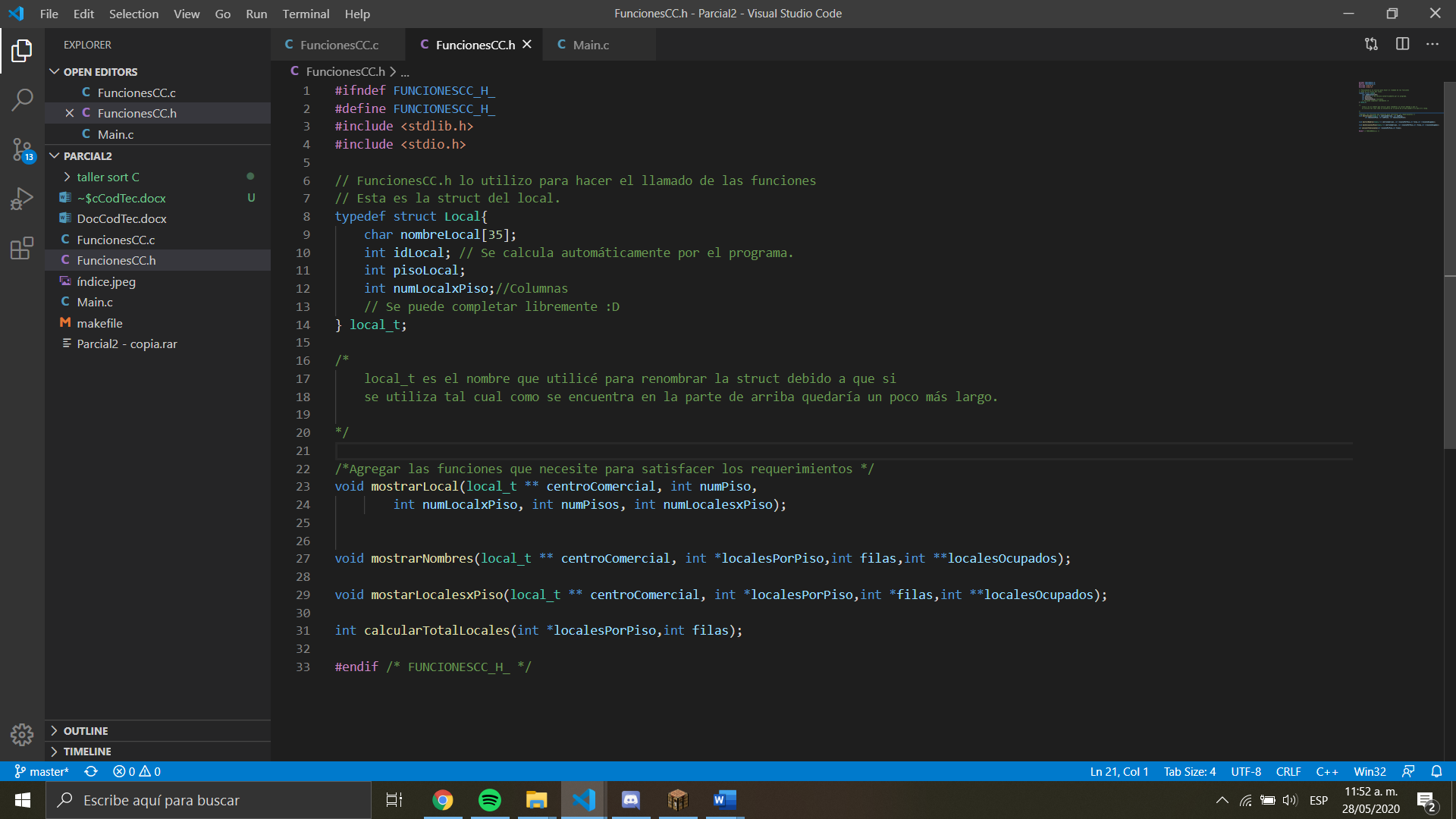
                break;

Caso 7 utiliza una función que anteriormente expliqué.

**ENTREGA MEJORADA:**

**1 Entrega:**

En la primer en la struct que había no le agregué nada porque sinceramente no lo pensé pero al hacer la sustentación me di cuenta que si podía agregarle más valores para que el programa que realicé fuera más comodo al utilizar.

****

**2. Entrega:**

**Funciones agregadas y más:**

* Se agregaron más atributos a la estructura local para hacer los respectivos ordenamientos, se añadió el **estadoActual**, el cual es de tipo **enum** e indica si el local está **abierto o cerrado**, la **capacidadLocal** que indica la **capacidad de un local (es asignado de forma aleatorio en el main)**, la **cantidadProductos** que es la **cantidad de productos en inventario**, las **ventasSemanales**, un entero para representar cuantos artículos venden por semana (es asignado de forma aleatoria en el main).
* En el Main cuando se selecciona la opción 2 para ingresar un nuevo local se imprime una matriz de 0s y 1s para que el usuario pueda ver cuales posiciones están libres y cuales ocupadas. -las nuevas funciones van desde la opción 8 hasta la 14.
* **La opción 8** abre un local que ingrese el usuario (fila y columna), es decir, cambia el estadoActual del local.
* **La opción 9** guarda la información actual del centro comercial en tres archivos distintos, dos de ellos son binarios (info.txt y localesOc.txt), el archivo tamano.txt guarda la cantidad de pisos y cuantos locales hay por piso para poder armar la matriz de nuevo cuando se carguen los datos. en info.txt se guarda la matriz del centro comercial y en localesOc.txt se guarda la información de la matriz localesOcupados para saber cuáles están libres y cuáles no.
* **La opción 10** carga la información de los archivos mencionados en el punto anterior y vuelve a armar la matriz del centro comercial, la de los locales ocupados y cuantos locales hay por piso.
* **La opción 11** ordena la matriz por la capacidad de cada local por piso, es decir, toma el piso 1 y ordena todos los locales ocupados de menor a mayor y así por cada piso (con selection sort).
* **La opción 12** ordena todos los locales ocupados de acuerdo con la cantidad de productos en inventario de mayor a menor usando insertion sort.
* **La opción 13** ordena todos los locales ocupados de acuerdo con la cantidad de ventas semanales que tengan usando merge sort.
* **La opcion 14** es similar a la 11, solo que no es por piso sino con el total de locales ocupados, y ordena de mayor a menor la capacidad de todos los locales ocupados usando quick sort.

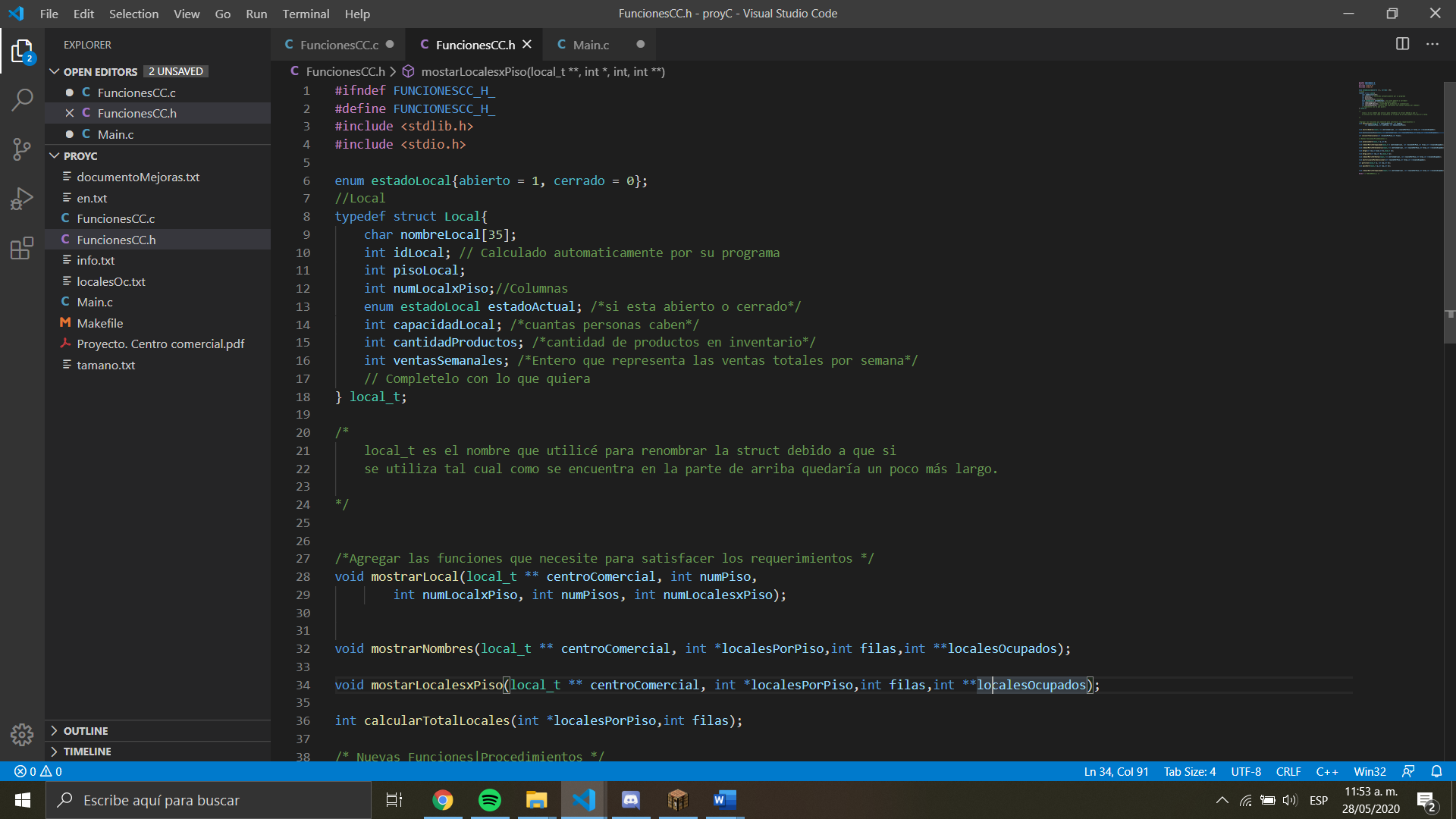
**Información extra en el struct: en total el struct que representa cada local del centro comercial debe tener al menos seis campos: de los cuáles al menos 1 debe ser un enum y otro debe ser numérico.**

Agregué otro enum con dos valores 0 y 1 para establecer un local como **Abierto [1]** o **Cerrado [2]**

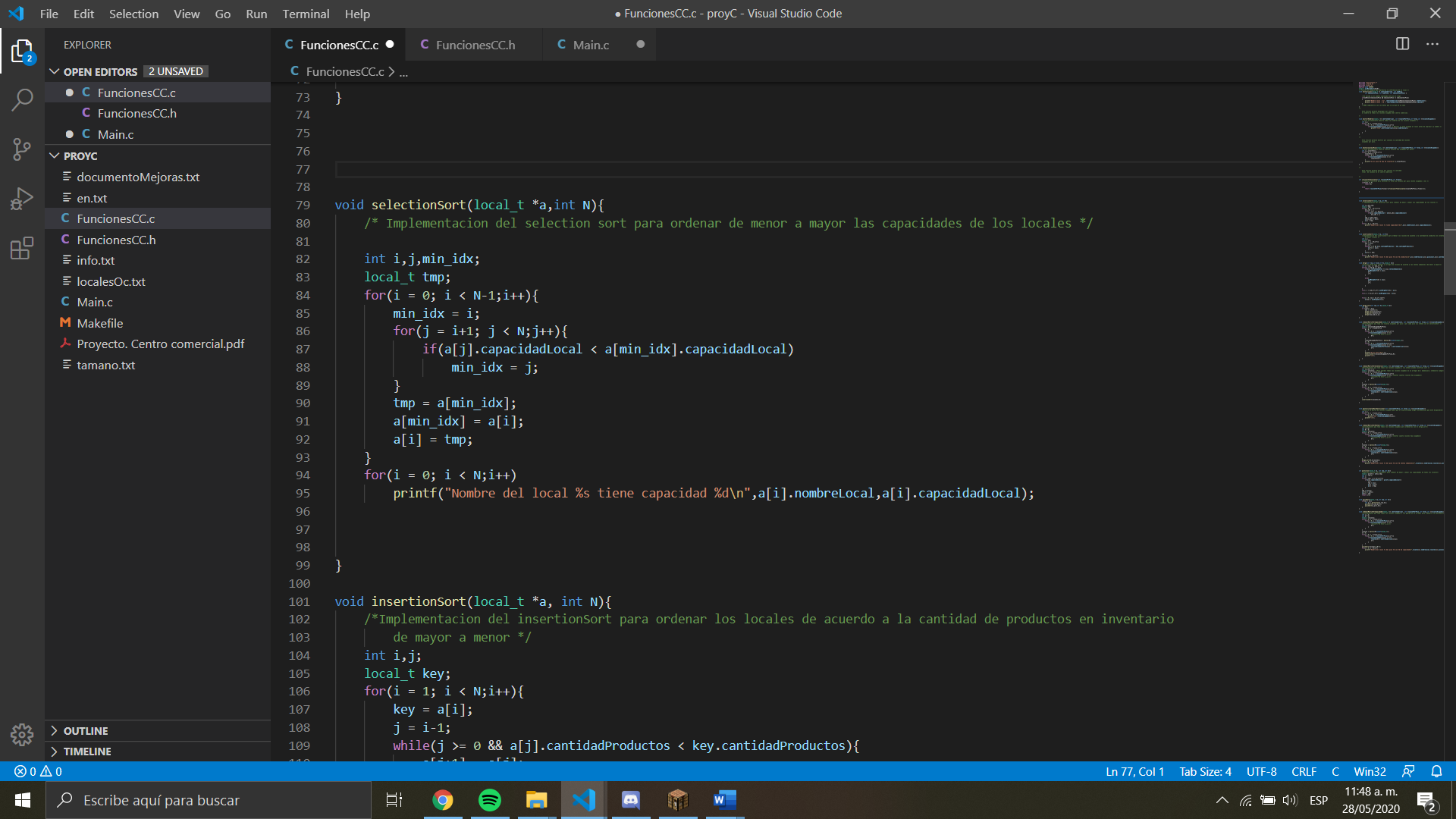
En la segunda entrega al saber que ya podía agregar más valores, implementé los siguientes

* Enum estadoLocal estadoActual -> Como lo comenté en el código lo que hace es colocarle un estado al local ya sea abierto o cerrado.
* capacidadLocal -> Para establecer cuántas personas caben en el local.
* cantidadProductos -> Para establecer la cantidad de productos en el inventario.
* ventasSemanales -> Es un entero para representar las ventas por semana.

**FOTO DE LA STRUCT NUEVA:**

****

**NUEVAS FUNCIONES|PROCEDIMIENTOS**

****

**Entradas:**

**int N -> Valor del arreglo**

**min\_idx -> Mínimo valor del arreglo.**

**Salidas:** Al hacer todo el anterior procedimiento saldrían en orden de menor a mayor las capacidades de los locales.

**Conceptos usados:** Ciclos, matrices, procedimientos, punteros y struct.

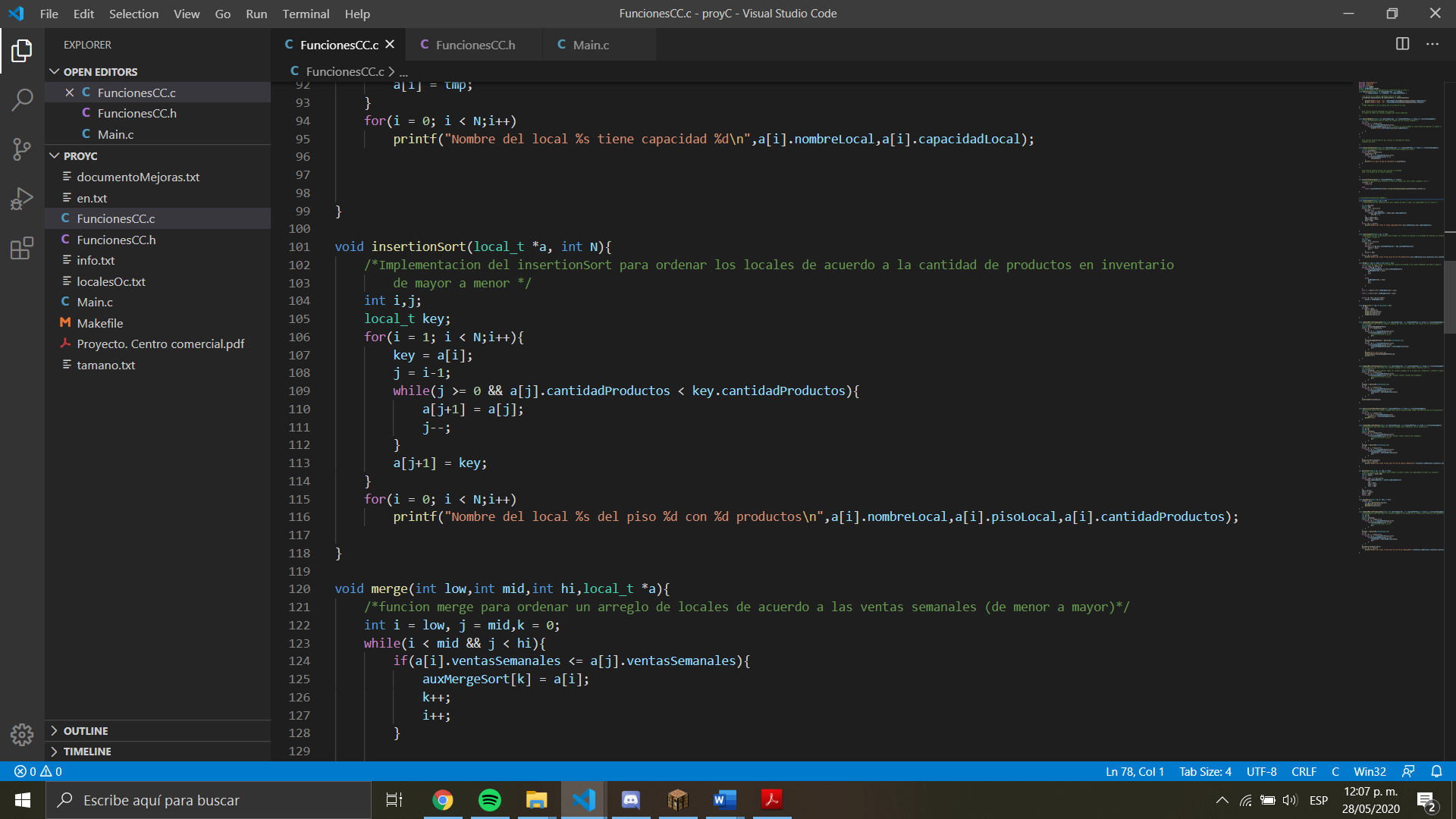
* **Ordenamiento iterativo de toda la matriz. Cree dos funcionalidades para ordenar la matriz que representa el centro comercial. Una de sus operaciones debe usar el algoritmo de selección y otra debe usar el algoritmo de inserción.**

**InsertionSort -> Ordenamiento Iterativo**

**¿Cómo Funciona?**

* Intuitivamente el algoritmo funciona retirando un elemento de la lista de elementos y volviéndolo a poner en la lista, pero esta vez en su posición correcta con respecto a una lista que ya se encuentra ordenada.

En este procedimiento lo que realiza es ordenar los locales de acuerdo con la cantidad de productos que tengan en el inventario de menor a mayor.

****

**El“Key” sirve para guardar el elemento que se va a ordenar, porque lo que hace el algoritmo es correr los elementos hacia la derecha para separar el espacio donde debe ir ubicado “Key”.**

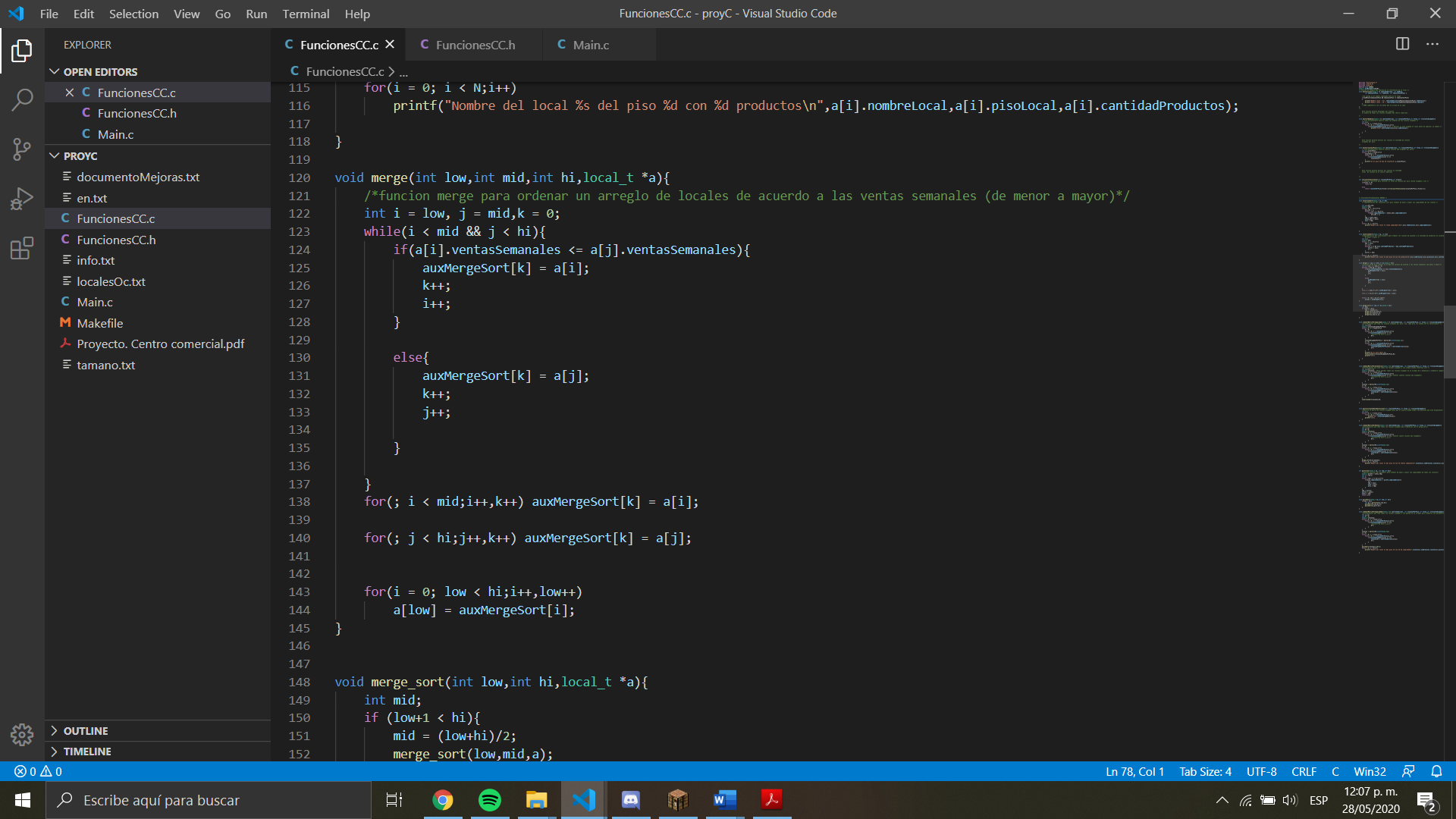
* **Ordenamiento recursivo. Cree funcionalidades para ordenar usando merge sort y quick sort los pisos de la matriz. Una funcionalidad para cada tipo de algoritmo. Por ejemplo, podría ordenar el piso por número de empleados, por valor de las ventas semanales, por cantidad de productos en el inventario, etc.**

**Merge y MergeSort -> Ordenamiento Recursivo.**

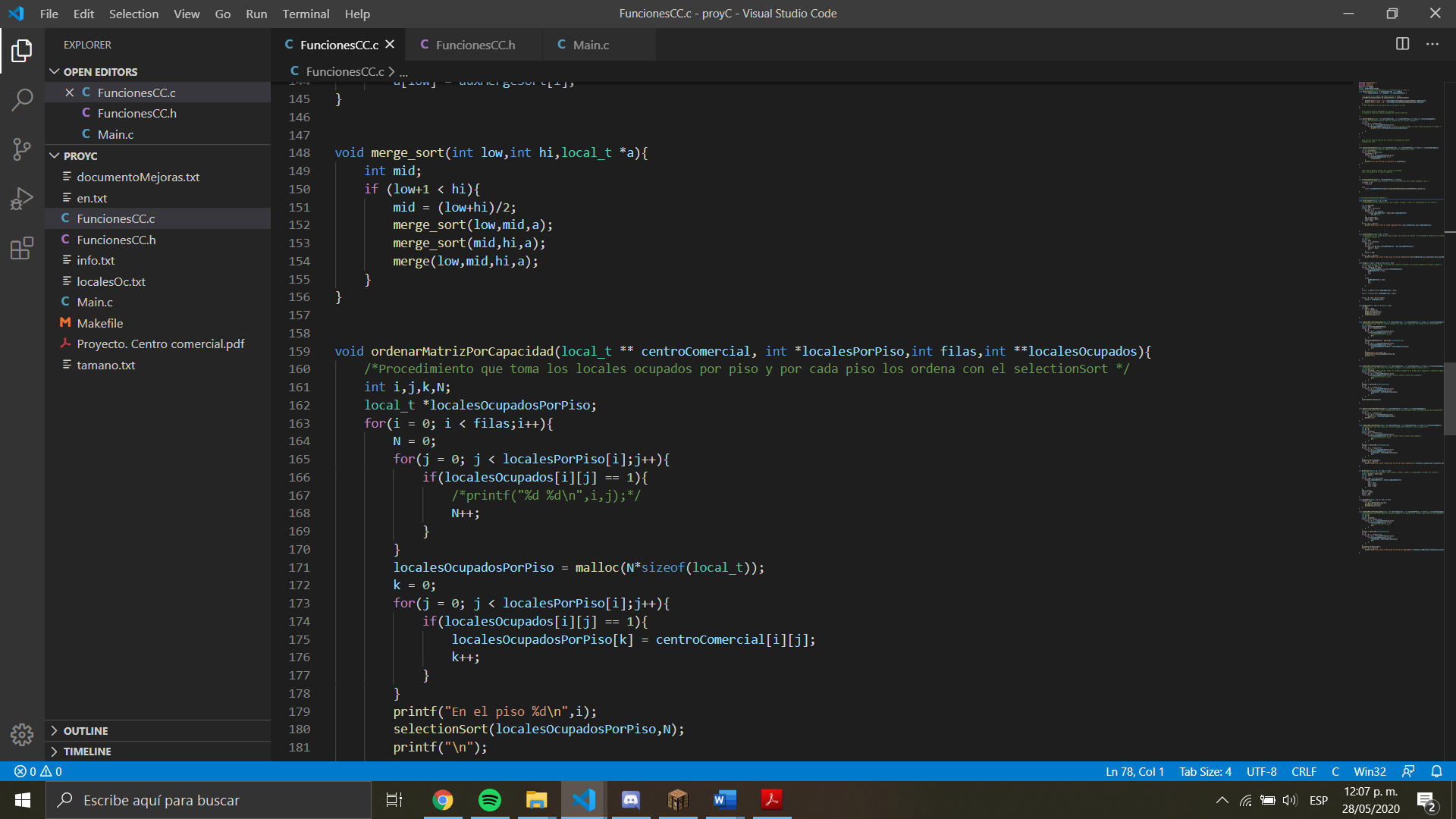
**¿Cómo funciona?**

* Conceptualmente, el ordenamiento por mezcla funciona de la siguiente manera:
* Si la longitud de la lista es 0 ó 1, entonces ya está ordenada.
* En otro caso: Dividir la lista desordenada en dos sublistas de aproximadamente la mitad del tamaño.
* Ordenar cada sublista recursivamente aplicando el ordenamiento por mezcla.
* Mezclar las dos sublistas en una sola lista ordenada.

En esta función merge lo que hace es ordenar un arreglo de locales de acuerdo con las ventas semanales [Menor a Mayor]

****

**Se explica en el código**

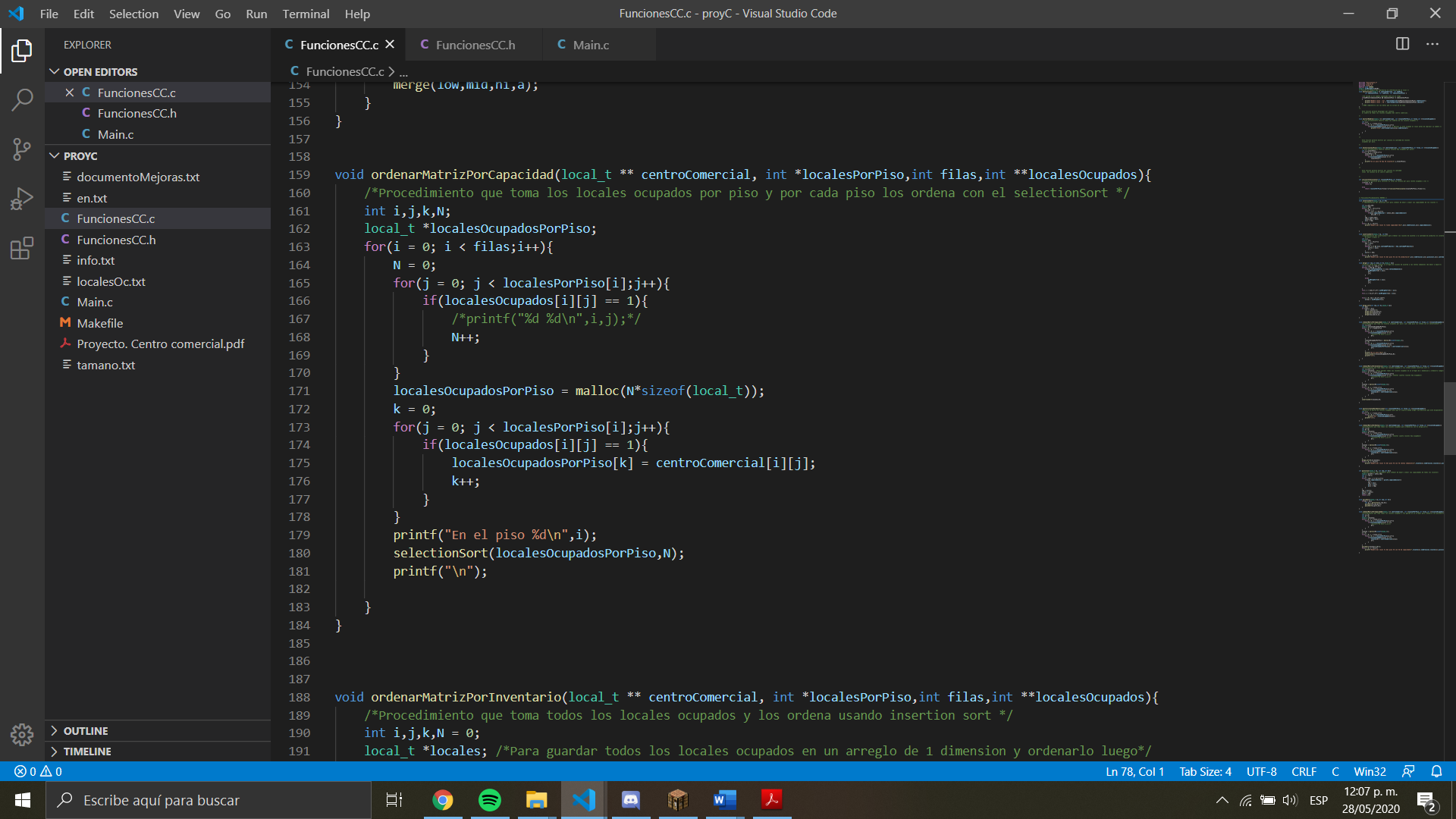
****

**Low -> Es un puntero que indica donde se quiere empezar a ordenar el arreglo.**

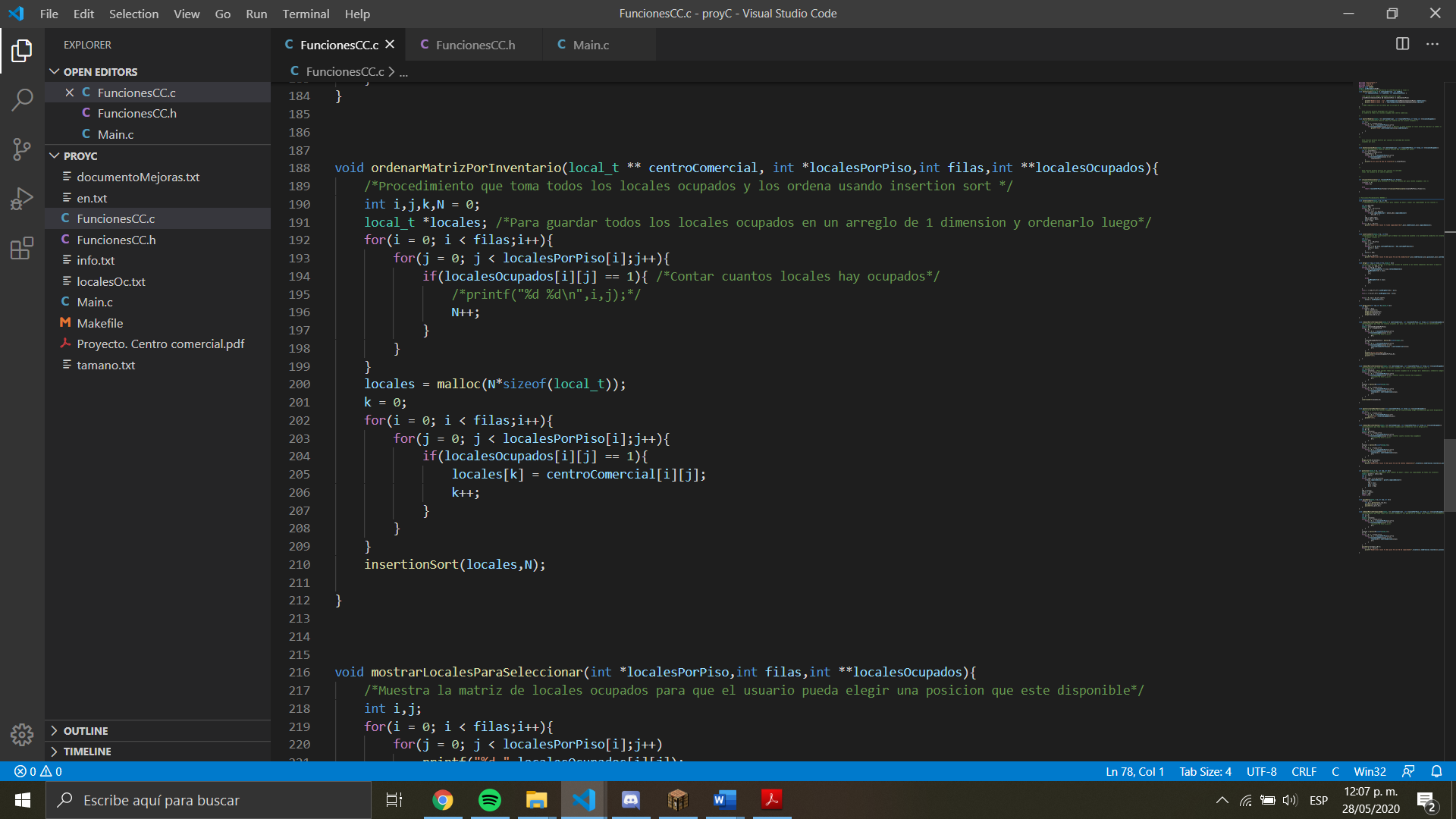
**Hi -> Es el puntero que indica hasta donde se quiere ordenar el arreglo.**

**Mid -> Es para partir el tamaño del arreglo a la mitad.**

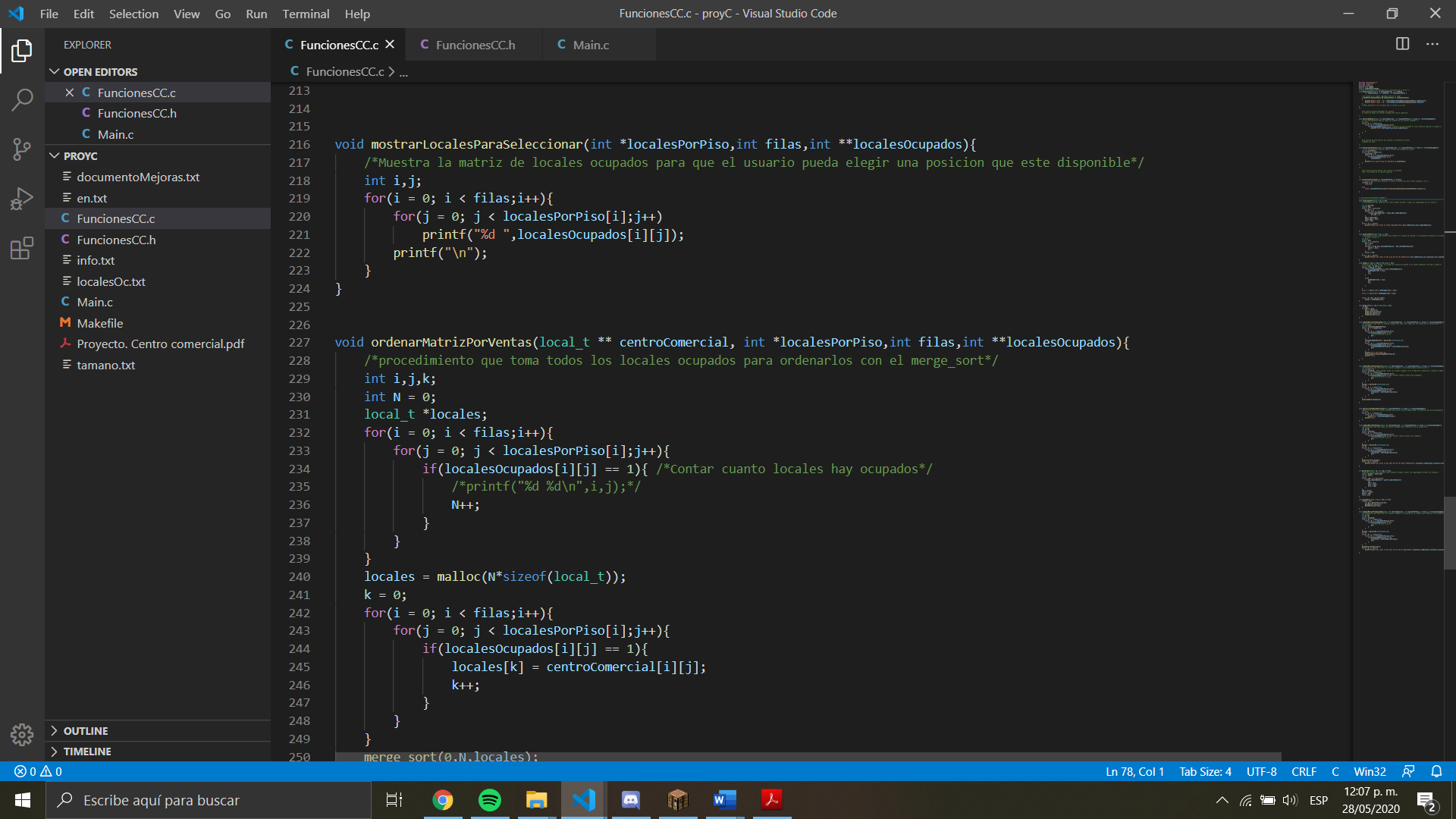
**Ejm: Si el arreglo es de tamaño 6, mid sería 3**

****

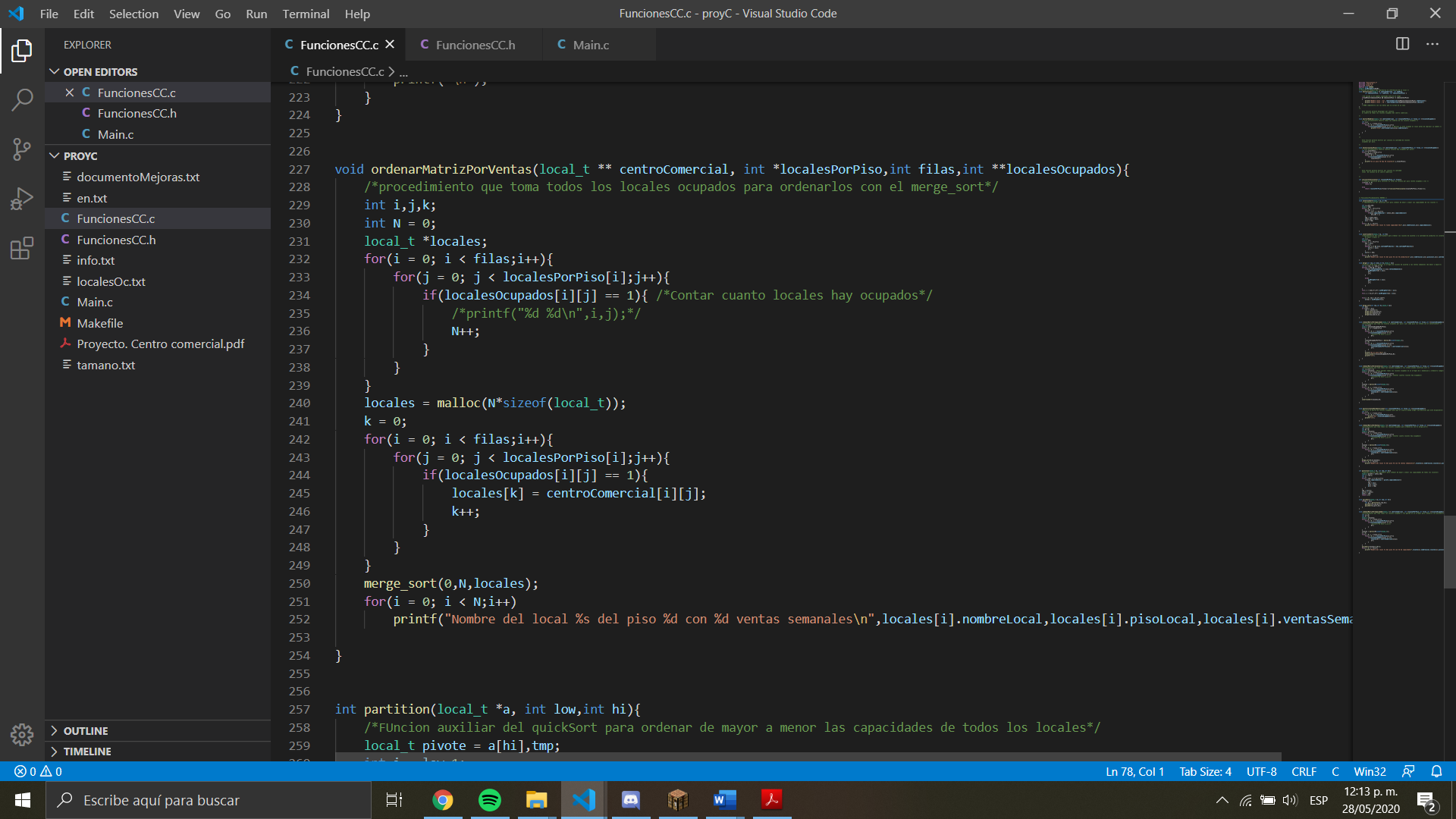
**Se explica en el código**

****

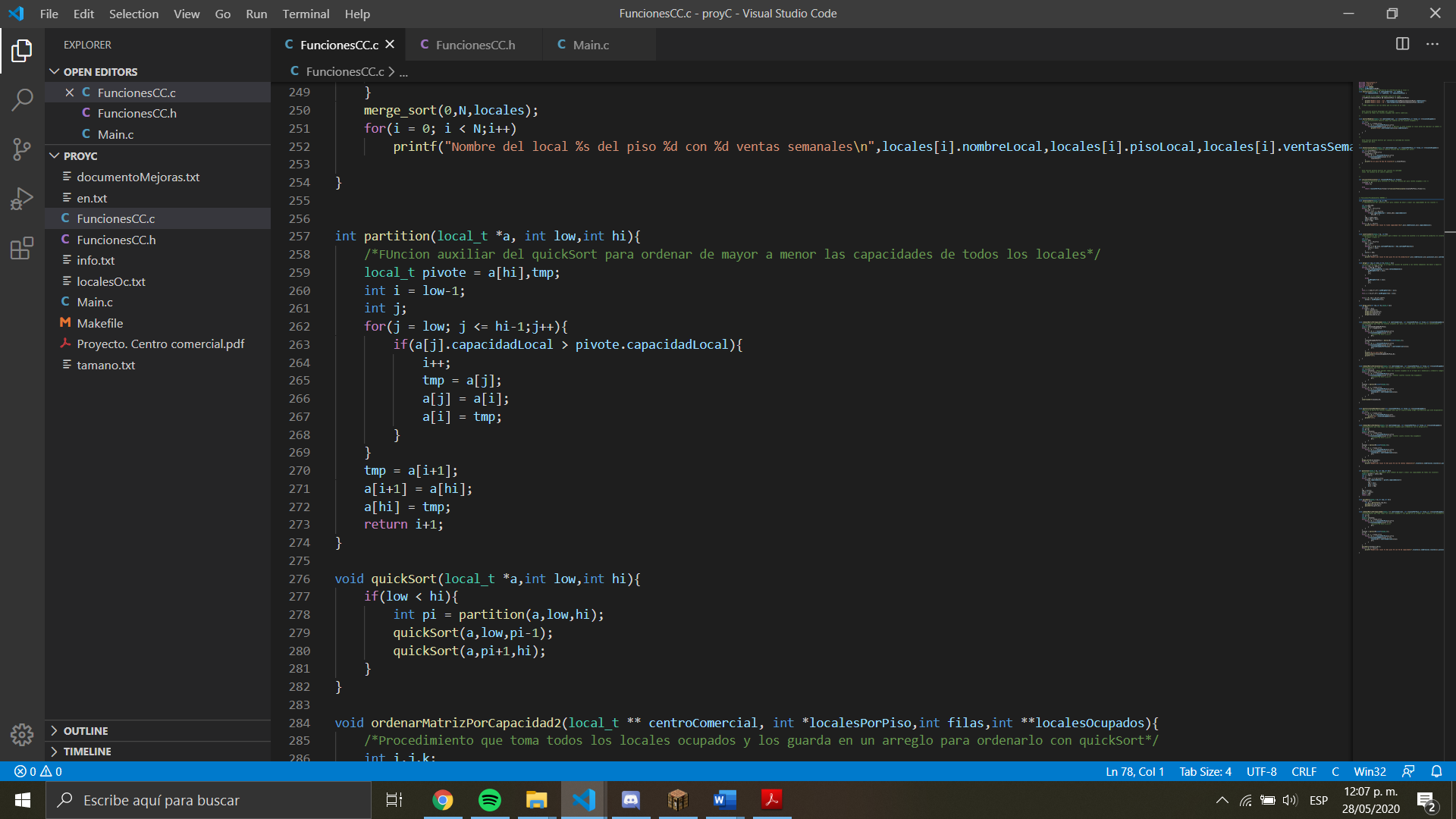
**Se explica en el código**

****

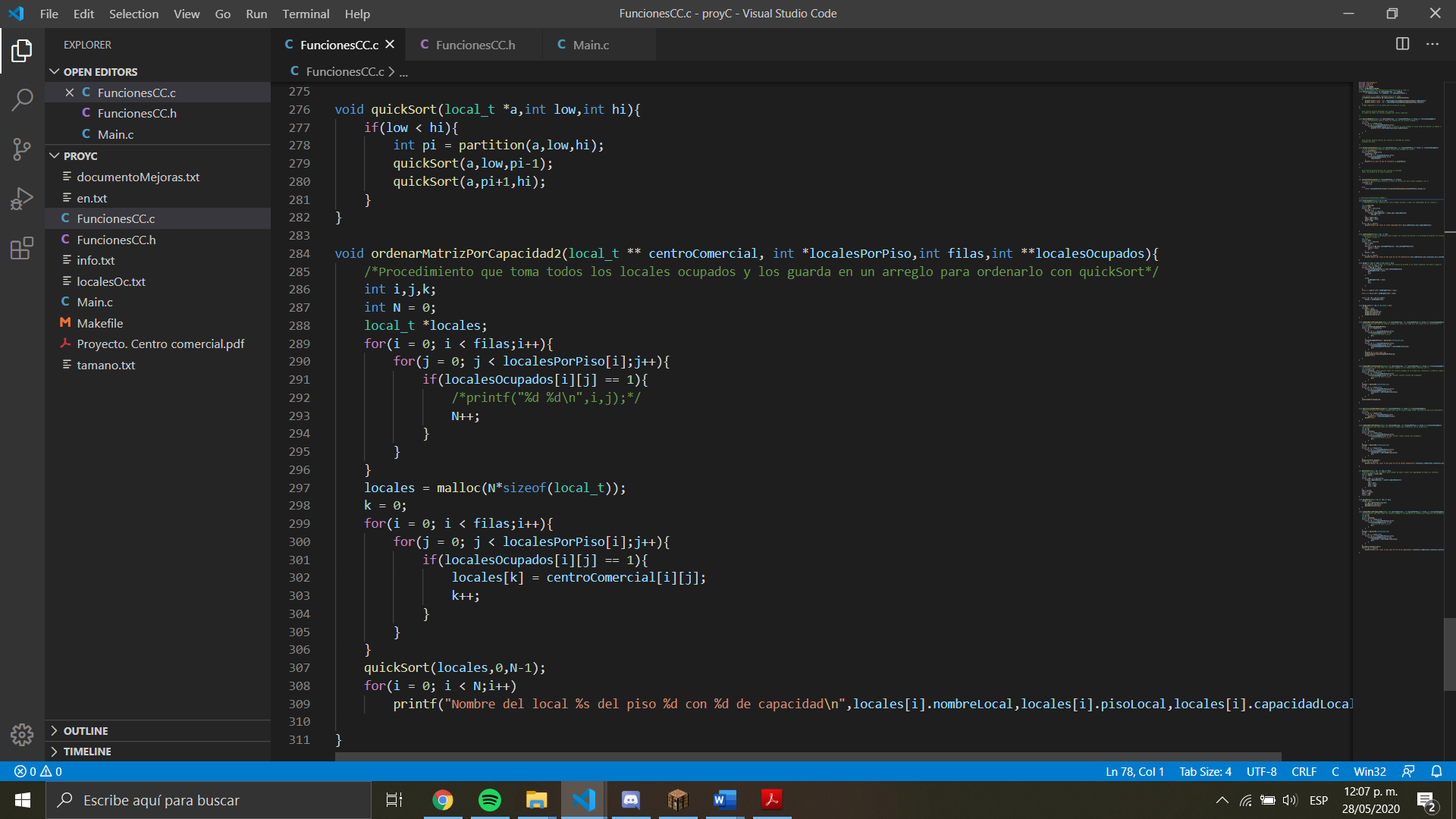
**Se explica en el código**

****

**Se explica en el código**

****

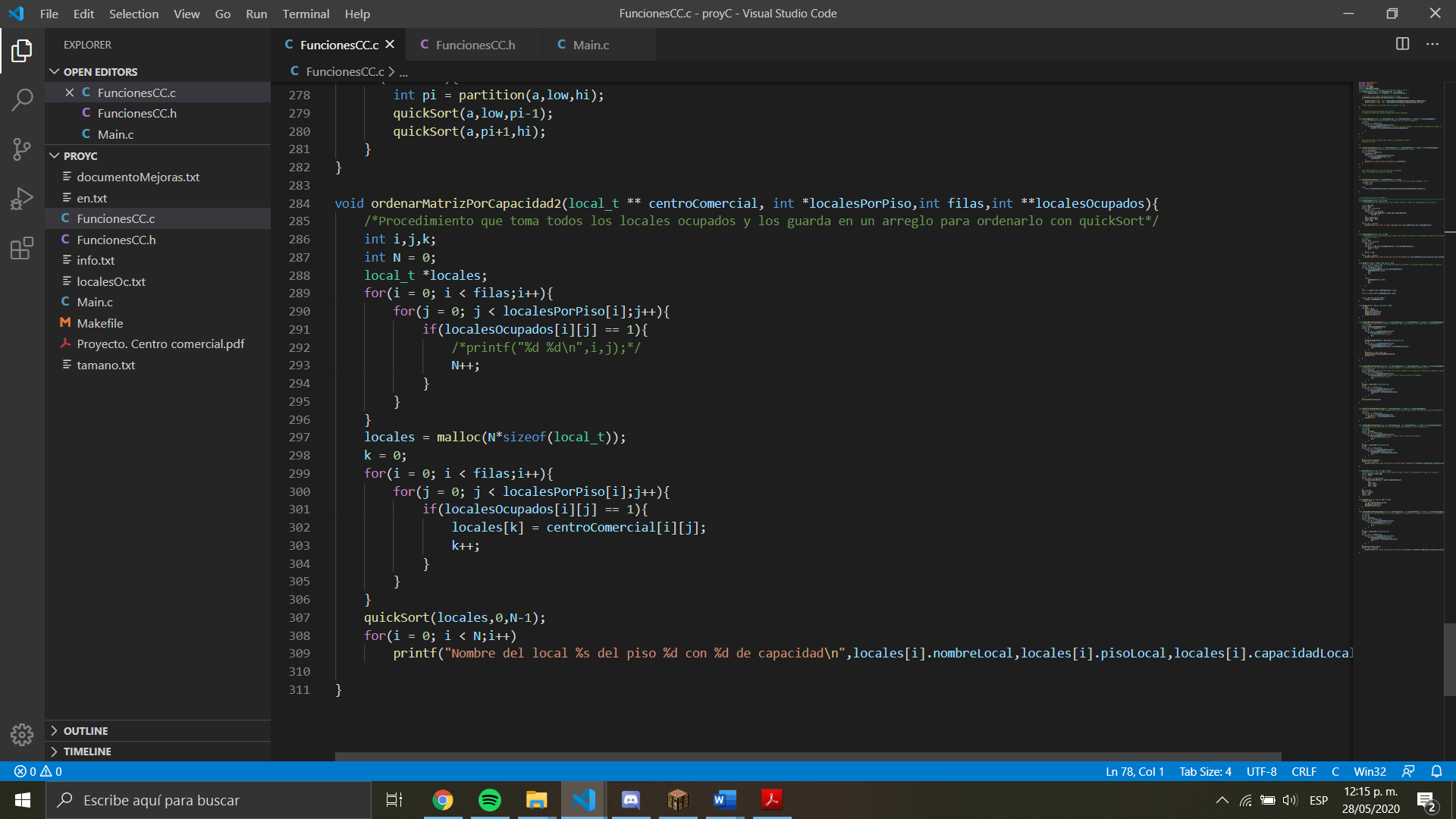
**Se explica en el código**

****

**Low -> Es un puntero que indica donde se quiere empezar a ordenar el arreglo.**

**Hi -> Es el puntero que indica hasta donde se quiere ordenar el arreglo.**

**Pi -> Es el partition index, índice en donde se va a partir el arreglo, no necesariamente como en el merge sort.**

****

**Se explica en el código**