Johann Emilson Ruano Perez

Proyecto final técnicas de programación

Cambios en las funciones:
Con respecto a la entrega anterior modifique bastante el programa, empezando desde la función ingresarLocal, cambié la condición para que el usuario ingresara el piso y el local en los cuales quería ubicar su negocio.

Versión parcial 2:

```
while(1){
```

De esta condición no tenia muy clara la idea de su funcionamiento.

Versión proyecto final:

```
}while ( ( piso > numPiso - 1 || local > numLocal - 1 ) || ( matrizMall [ piso ][ local ].status == OCUPADO ) );
```

Con esta condición el programa verifica que los datos ingresados por el usuario no estén fuera del rango de la matriz y también que el espacio no este en uso por otro negocio.

En esta función también cambie la forma de asignarle los datos ingresados por el usuario a las variables.

Versión parcial 2:

```
printf( "Cual es el nombre del local?:\n" );
scanf( "%s", punt-> nombreLocal );
punt->idLocal = idLocal;
punt->idLocal = rand () % (N-M+1) + M;
punt->status = 1;
```

Aquí asignaba los valores con una variable punt -> la cual nunca había declarado y eso me generaba problemas al compilarlo

Versión proyecto final:

```
matrizMall[ piso ][ local ].numEmpleados = nEmpleados;
matrizMall[ piso ][ local ].empleadosEnTurno = nEmTurno;
matrizMall[ piso ][ local ].elementosInventario = nInventario;
matrizMall[ piso ][ local ].status = OCUPADO;
matrizMall[ piso ][ local ].idLocal = rand() % 10000;
```

Aquí asigno los valores en la matriz con las posiciones ingresadas por el usuario, esta fue una retroalimentación que me dio la profesora y con la cual comprendí que de esta manera es mas fácil y mas eficaz en comparación a la anterior que había hecho.

En la función listarLocales se le agrego un condicional if, el cual se encarga de verificar que los locales esten ocupados por un negocio, esto se hace para que no se impriman casillas vacias las cuales llenaran espacio en pantalla.

Versión parcial 2:

```
for( int i = 0; i < numPiso; i++){
    for( int j = 0; j < numLocal; i++ ){
        printf( "%s, %d, %d, %d", matrizMall[ i ][ j ].nombreLocal, matrizMall[ i ][ j ].idLocal,
        matrizMall[ i ][ j ].pisoLocal, matrizMall[ i ][ j ].numLocalxPiso );
```

Aquí imprime todo independientemente que esa casilla tenga datos o no.

Versión proyecto final:

```
for( i = 0; i < numPiso; i ++ ){
  for( j = 0; j < numPiso; j ++ ){
    if ( matrizMall[ i ][ j ].status == OCUPADO ){/
```

Aquí me aseguro de que el local este ocupado para poder imprimir sus datos.

En la función buscarPorNombre funciona igual que la versión anterior, solo se agregaron unos nuevos prints que enseñaran en pantalla los nuevos datos agregados al struct local_t.

En la función buscarPorId al igual que en la función anterior solo le agregaron los prints para mostrar los nuevos datos asociados al local.

La función cambiarNombre hace las comparaciones igual tanto en el parcial 2 como en el proyecto final, lo que cambio fue la manera de asignarle el nuevo nombre al local.

Versión parcial 2:

```
printf( "Escribe el nuevo nombre:\n " );
scanf( "%s", newName );
strcpy( punt->nombreLocal, newName ) ;
```

Aquí pasa lo mismo que en ingresarLocal, pues asignaba el nombre con un puntero que jamás había sido declarado y por esta razón el programa no compilaba.

Versión proyecto final:

```
printf( "Escribe el nuevo nombre del local\n" );
scanf( "%s", matrizMall[ i ][ j ].nombreLocal );
printf( "El nombre ha sido cambiado con exito\n" );
```

Aquí hice lo mismo que recomendó la profesora, poner la matriz con sus índices e indicar que ese nombre ingresado por el usuario se guardaría en el struct del local.

Función eliminarLocal, le cambie la condición para asegurarme de que los datos ingresados estuvieran dentro de la matriz.

Versión parcial 2:

```
while(1){
```

Al igual que ingresarLocal utilizaba este while el cual no entendía muy bien como funcionaba.

Versión proyecto final:

```
}while( ( pis > numPiso ) || ( loc > numLocal ) );
```

Con este while me aseguro de que los datos que estén ingresados por el usuario no sobrepasen el limite de la matriz.

Nuevos elementos en el código:

Agregué dos funciones de conteo, localesLibres y localesOcupados, la primera se encargará de recorrer toda la matriz y deberá contar cuantos locales tienen la constante DISPONIBLE, finalmente esta nos mostrara en pantalla cuantos locales están libres para poder usarlos. La función localesOcupados hará exactamente lo mismo, pero esta contará los locales que ya están en uso.

Agregué 4 funciones de ordenamiento, usando los siguientes tipos de ordenamiento: selection sort, merge sort, insertion sort y quick sort, todas estas funciones imprimirán los números ordenados de menor a mayor, pero usando un algoritmo diferente.

La primera es ordenarPorNumEmpleados donde se hace uso del algoritmo Selection sort, en esta función el usuario digitará que piso deseará ordenar, después de esto se imprimirá una lista con los valores ordenados del numero de empleados por local.

La segunda es ordenarPorVentas donde se hace uso del algoritmo Merge sort, al igual que en la función anterior el usuario digitará que piso deseará ordenar, después de esto se imprimirá una lista con los valores ordenados de la cantidad de ventas semanales por local.

La tercera es ordenarPorInventario donde se hace uso del algoritmo Insertion sort, al igual que en las funciones anteriores, el usuario digitará que piso deseará ordenar, después de esto se imprimirá una lista con los valores ordenados de la cantidad de elementos en el inventario por local.

La cuarta es ordenarPorEmpleadosTurno donde se hace uso del algoritmo Quick sort, al igual que en las funciones anteriores, el usuario digitará que piso deseará ordenar, después de esto se imprimirá una lista con los valores ordenados de la cantidad de empleados en su turno laboral.

Otra función nueva es guardarCC la cual se encarga de guardar la matriz y sus datos en un archivo .dat.

También hice un nuevo struct, el cual representa de que categoría es el local, poder ser de tipo hogar, comida, tecnología, vestuario.

```
typedef enum category{
HOGAR,
COMIDA,
TECNOLOGIA,
VESTUARIO,
}category_l;
```

Como ya lo había mencionado antes, también agregue algunos nuevos elementos dentro del struct local, los cuales utilice para hacer los ordenamientos.

```
typedef struct Local{
        char nombreLocal[ 35 ];// Nombre del local
        int idLocal; // Calculado automaticamente por su programa
        int pisoLocal; // Piso donde esta ubicado el local
        int numLocalxPiso;//Columnas
        int numEmpleados; // Cantidad de empleados que tiene el local
        int ventasSemanales;// Ventas que ha hecho el local en una semana
        int elementosInventario;//Candidad de elementos en el inventario del local
        int empleadosEnTurno;// Empleados que estan trabajando actualmente dentro del local
        status_l status;// Estado del local
        category_l category;// Categoria del local
        // Completelo con lo que quiera
} local_t;
```

Aquí incorporo el nuevo struct y los nuevos datos dentro de local.

Por último, en Main.c cuando la matriz era creada con las medidas digitadas por el usuario, además de ser creada también se imprime, eso lo hice para asegurarme de que se estuviera creando con las dimensiones correctas.

```
for( i = 0; i < numPiso; i++ ){
  for( j = 0; j < numLocal; j++ ){
    printf("%d", matrizMall[ i ][ j ].status );
  }
  printf( "\n" );
}
```

```
Ingrese la cantidad de pisos del centro comercial:
6
Ingrese la cantidad de locales del centro comercial:
5
00000
00000
00000
00000
00000
00000
```