```
function kiosco(){}
function almacen(){}
function restaurante(){}
function jugueteria(){}
procedure AgruparTiendasHogareñasBajoColor_(color){
 /*  Proposito: Agrupar en cada sector del centro comercial las tiendas hogareñas
            bajo el color *color* y si no hay tiendas hogareñas convertir
            el sector en un shopping
       Precondición: Ninguna
       Parametros: *Color* - color - representa el color bajo el que se agrupan
              las tiendas hogareñas.
       Observación: es un recorrido de procesamiento sobre los sectores del centro
            comercial, agrupando tiendas hogareñas o convirtiendo en shopping
            - las tiendas hogareñas son los kioscos y los almacenes  */
   IrAlInicioDeUnRecorridoAl_Y_(Este, Norte)
   AgruparTiendasHogareñasBajoColor_OConvertirEnShopping(color)
   while(haySiguienteCeldaEnUnRecorridoAl_Y_(Este, Norte)){
      IrASiguienteCeldaEnUnRecorridoAl_Y_(Este, Norte)
      AgruparTiendasHogareñasBajoColor_OConvertirEnShopping(color)
   }
}
procedure AgruparTiendasHogareñasBajoColor_OConvertirEnShopping(color){
   /*    Proposito: Agrupar en el sector actual las tiendas hogareñas
            bajo el color *color* y si no hay tiendas hogareñas convertir en un shopping
       Precondición: Ninguna
       Parametros: *color* - color - representa el color bajo el que se agrupan
              las tiendas hogareñas.
       Observación: las tiendas hogareñas son los kioscos y los almacenes    */
   if(hayTiendasHogareñasAcá()){
      AgruparTiendasHogareñasBajoColor_Acá(color)
   }else{
```

```
      ConvertirEnShopping()
   }
}
function hayTiendasHogareñasAcá(){
   /*   Proposito: Indica si en el sector actual hay al menos un kiosco o un almacen
   Precondicion: ninguna
   Tipo: Booleano   */
   return (hayBolitas(kiosco()) || hayBolitas(almacen()))
}
procedure AgruparTiendasHogareñasBajoColor_Acá(color){
   /*   Proposito: Agrupar en el sector actual las tiendas hogareñas
           bajo el color *color*
   Precondicion: Tiene que existir al menos una tienda hogareña en el sector actual
   Parametros: *color* - color - representa el color bajo el que se agrupan
              las tiendas hogareñas.   */
   cantTiendasHogareñas := cantTiendasHogareñasAca()
   VaciarCelda()
   Poner_DeColor_(cantTiendasHogareñas, color)
}
function cantTiendasHogareñasAca(){
   /*   Proposito: Describe la cantidad de tiendas hogareñas en el sector actual
   Precondicon: ninguna
   Tipo: Número   */
   return (nroBolitas(kiosco())+ nroBolitas(almacen()))
}
procedure ConvertirEnShopping(){
   /*   Proposito: Convierte en shopping el sector actual
   Precondicion: No debe existir tiendas hogareñas en el sector actual   */
   VaciarCelda()
   Poner(Verde)
}
```

```
//---------------------------------------- Ejercicio 2 ----------------------------------------------

   procedure IrACentroDeVentasEnColumnaDeTiendaDe_(clase){

   /*    Proposito: Ubica el cabezal en el centro de ventas de las tiendas de la clase *clase*

         de la columna actual

   Precondicion: Hay un centro de ventas de las tiendas del tipo dado en la columna actual

   Parametro: *clase* -color - describe el color de la clase de tienda a buscar su centro de
ventas

   Observación: -un centro de ventas es el sector que agrupa la mayor cantidad de tiendas

            del tipo dado en la columna, agrupando la cantidad de tiendas de si mismo

            y las de los sectores lindantes en las 4 direcciones ortogonales y si es que hubiese

            - es un recorrido de busqueda sobre la columna actual, buscando el centro de venta
de la clase dada    */

   IrAlBorde(Sur)

   while(not esCentroDeVentaDeTiendaDe_(clase)){

      Mover(Norte)

   }

}

function esCentroDeVentaDeTiendaDe_(clase){

   /*    Proposito: Indica si el sector actual es el centro de ventas de tiendas de la clase *clase*

   Precondicion: Ninguna

   Parametro: *clase* -color - describe el color de la clase de tienda a buscar su centro de
ventas

   tipo: Booleano    */

   return (cantTiendasAgrupadasDe_Aca(clase) ==
cantTiendasDeCentroDeVetasDe_EnColumna(clase))

}

function cantTiendasAgrupadasDe_Aca(clase){

   /*    Proposito: Describe la cantidad de tiendas agrupadas de la clase *clase* en el sector
actual

   Precondicion:Ninguna

   Parametro:*clase* -color - describe el color de la clase de tienda a buscar su centro de
ventas
```

```
   tipo: Número
   Observación: es un recorrido de acumulación sobre los sectores lindantes al sector actual  */
   cantTiendasVistas:= cantTiendasDe_Aca(clase)
   dirActual:= minDir()
   while(dirActual/=maxDir){
      cantTiendasVistas:= cantTiendasVistas + cantTiendasDe_Al_(clase, dirActual)
      dirActual:= siguiente(dirActual)
   }
   return(cantTiendasVistas + cantTiendasDe_Al_(clase, dirActual)
}
function cantTiendasDe_Aca(clase){
   /*    Proposito: Describe la cantidad de tiendas de clase *clase* en el sector actual
   Precondicion: Ninguna
   Parametro:*clase* -color - describe el color de la clase de tienda a buscar su centro de
ventas
   Tipo: Número    */
   return (nroBolitas(clase))
}
function cantTiendasDe_Al_(clase, direccion){
   /*    Proposito: Describe la cantidad de tiendas de clase *clase* en el sector lindante hacia la
direccion *direccion*,
         si no hay un sector lindante hacia la direccion dada describe 0
   Precondicion:NINGUNA
   Parametro:*clase* -Color - describe el color de la clase de tienda a buscar su centro de
ventas
         - *direccion* - Direccion - Direccion hacia donde sensar
   Tipo: Número    */
   return (choose cantTotalTiendasDe_Al_(clase, direccion) when (puedeMover(direccion))
               0 otherwise  )
}
```

```
function cantTotalTiendasDe_Al_(clase, direccion){
  /*
  Proposito: Describe la cantidad de tiendas de clase *clase* en el sector lindante hacia la direccion *direccion*
  Precondicion:Debe existir al menos un sector hacia la direccion dada
  Parametro:*clase* -Color - describe el color de la clase de tienda a buscar su centro de ventas
      - *direccion* - Direccion - Direccion hacia donde sensar
  Tipo: Número
  */
  Mover(direccion)
  return (cantTiendasDe_Aca(clase))
}
function cantTiendasDeCentroDeVetasDe_EnColumna(clase){
  /*
  Proposito: describe la cantidad de tiendas del centro de venta de tiendas de clase *clase* en la columna actual
  Precondicion: Debe existir al menos un centro de ventas de la clase dada en la columna actual
  Parametro:*clase* -Color - describe el color de la clase de tienda a buscar su centro de ventas
  Observación: Es un recorrido de maximo/minimo sobre la celda de la columna actual, buscando
        el centro de venta de tiendas de la clase dada
  */
  IrAlBorde(Sur)
  cantMaximaTiendasAgrupadas:= cantTiendasAgrupadasDe_Aca(clase)
  while(puedeMover(Norte)){
    Mover(Norte)
    cantMaximaTiendasAgrupadas:= maximoEntre_Y_(cantMaximaTiendasAgrupadas, cantTiendasAgrupadasDe_Aca(clase))
  }
  return (cantMaximaTiendasAgrupadas)
}
```

```
procedure IrANésimaVacía_(nnn) {
   /*    Proposito
      Mueve el cabezal a la celda vacia numero *n*, si no hay suficientes celdas vacias
      deja el cabezal en el borde Noreste
   Precondicion
      ninguna
   Observaciones
      es es un reccorido de busqueda sobre las celdas del tablero, que busca la celda
      vacia numero *n*    */
   celdasVacíasYaVistas := 0
   IrAPrimeraCeldaEnUnRecorridoAl_Y_(Este,Norte)
   celdasVacíasYaVistas := celdasVacíasYaVistas + unoSi_CeroSino(esCeldaVacía())
   while(haySiguienteCeldaEnUnRecorridoAl_Y_(Este,Norte) && celdasVacíasYaVistas /= nnn){
   IrASiguienteCeldaEnUnRecorridoAl_Y_(Este,Norte)
   celdasVacíasYaVistas := celdasVacíasYaVistas + unoSi_CeroSino(esCeldaVacía())
   }
}
```