**PROYECTO INFORMATICA 2**

* **Análisis del problema:**

Se requiere hacer un programa que reciba como entrada un arreglo unidimensional (regla) la cual tiene números en la primera(A) y segunda posición(B) que se refieren a las “coordenadas” del número que se quiere comparar de un arreglo bidimensional determinado por estos dos primeros números(A,B), por lo que primero se buscará el número mayor de estos dos, luego se determinará si es un número par o impar, si resulta ser un número impar(A), entonces la dimensión del primer arreglo bidimensional resultara ser de orden AxA, pero si el número mayor resulta ser par(B), entonces la dimensión del arreglo será de orden BxB. Luego de identificar el orden del primer arreglo (base), se tomarán los valores de la “regla” que estén en la posición siguiente a los números A y B que habíamos usado antes y los llamaremos “C”, con ello se buscará crear arreglos bidimensionales de dimensiones cuadrados que puedan ser alineados unos detrás de otros, a través de un punto llamado “centro”. Para estos nuevos arreglos que se irán creando se deberán cumplir ciertas condiciones dependiendo del valor de “C”, estos valores pueden ser:

1. (C = -1): esto quiere decir en primera instancia (partiendo desde el arreglo “base”) que el número(N) que este en la ubicación A, B debe ser menor al número(Z) de un arreglo bidimensional que esté alineado con nuestro arreglo Base a través del centro, una vez alineados los centros de los dos arreglos, se hará la comparación entre el número Z y el número del nuevo arreglo que esté alineado directamente. Si no se cumple la condición, se deberá “rotar” el nuevo arreglo tres veces más y si aún no se cumple la condición se deberá intentar los mismo con un arreglo diferente que sea 2 dimensiones mayor a la del arreglo que habíamos probado antes, así hasta que se cumpla la condición y al arreglo que cumpla esta condición lo llamaremos “base2”. Si se cumple la condición entonces se deberá tomar otro valor a C y dependiendo de su valor hacer la respectiva comparación con un número de un arreglo bidimensional nuevo comenzando con el mismo orden del arreglo Base y aumentando de 2 en 2 las dimensiones, así hasta que C sea el último valor de la “regla”.
2. (C = 0): partiendo del arreglo “base” , se busca crear un arreglo bidimensional (base2) cuadrado que se pueda alinear su “centro” con el “centro” de nuestro arreglo “base”, para luego ubicarnos en el número que se encuentre en la posición A,B y hacer la comparación con el número que esté alineado del nuevo arreglo (base2) a la posición A,B de nuestra arreglo “base”, una vez determinados los dos números a comparar, se debe cumplir condición de que el número en nuestro arreglo “Base2” debe ser igual al número alineado del nuevo arreglo (base2), si no se cumple la condición se deberá “rotar” 3 veces el nuevo arreglo(base2) hasta que se cumpla la condición, luego se deberá cambiar el valor de C a la siguiente posición de la “regla” y generar un nuevo arreglo bidimensional el cual debe partir desde la dimensión de la “Base” inicial e ir aumentando dimensiones de 2 en 2 hasta que cumpla la condición de C aplicando el mismo método ya descrito anteriormente hasta que C sea el último valor de la “regla”.
3. (C = 1): estando en el arreglo “Base” se busca alinear nuevos arreglos bidimensionales(base2) cuadrados que al alinear su “centro” con el “centro” del arreglo ”base”, se compare el número que se encuentre en la ubicación A,B del arreglo “Base” con el número que esté alineado del nuevo arreglo(base2) y se debe cumplir la condición de que el número del primer arreglo debe ser mayor al número que esté alineado del nuevo arreglo(base2), si no se cumple la condición se deberá “rotar” 3 veces el nuevo arreglo(base2) hasta que se cumpla la condición, luego se deberá cambiar el valor de C a la siguiente posición de la “regla” y generar un nuevo arreglo bidimensional el cual debe partir desde la dimensión de la “Base” inicial e ir aumentando dimensiones de 2 en 2 hasta que cumpla la condición de C aplicando el mismo método ya descrito anteriormente hasta que C sea el último valor de la “regla”.

Para la solución del problema se tiene previsto:

* + Pedir al usuario que ingrese la “regla” y eso se hará mediante una función que valide las entradas y se asegure que sean datos validos con los que se pueda trabajar.
  + Se obtendrá la posición del número ubicado en el primer arreglo usando los dos primeros datos de la “regla” y con esto se determinará el tamaño del primer arreglo bidimensional.
  + se hará una función para crear arreglos bidimensionales que contengan valores enteros comenzando con el número uno.
  + Luego de crear el primer arreglo y asegurar el valor del entero en la ubicación dada, se usarán los datos siguientes a los dos primeros de la “regla” para saber las condiciones de comparación, esto se hará iterando sobre esta regla cada vez que se encuentre un arreglo que cumpla con la condición dada (-1,0,1).
  + Se creará una función para “rotar” un arreglo en contra de las manecillas del reloj.
  + Se comparará el número del primer arreglo con un número del siguiente arreglo creado comenzando desde la misma dimensión del primer arreglo. si las dimensiones del primer arreglo y del siguiente a comparar son las mismas, entonces se comparará un numero en la misma posición de ambos arreglos y se “rotará” el ultimo arreglo hasta que se cumpla la condición (-1,0,1). Si no se cumple la condición incluso “rotando” el arreglo completo, se aumentará 2 veces la dimensión de este último y así mismo se aumentará dos veces las posiciones para comparar los enteros, pero ahora con este nuevo arreglo.
  + Se deberá guardar el orden del arreglo que cumpla con las condiciones dadas por la “regla”. Estas dimensiones estarán en un arreglo que lo llamaremos “cerradura”
* **Consideraciones para solución propuesta**

Se deberán utilizar arreglos dinámicos para facilitar la creación de las estructuras de datos bidimensionales de tamaño variable, además de que se debe validar correctamente la entrada de la “regla” para generar la certeza de trabajar con valores dentro de lo acordado y que el programa no genere inconvenientes, sumado a esto se debe tener presente que, al trabajar con memoria dinámica, también se debe ser responsable y saber “liberarla” cuando sea necesario.

* **Algoritmos implementados**
  + rotarM(int\*\*, int): esta función tiene como parámetro un doble puntero a un arreglo bidimensional de tipo entero y un entero que equivaldría a la dimensión del arreglo. Esta función retorna un doble puntero a un arreglo bidimensional que sería igual al arreglo original pero rotado 90 grados en contra de las manecillas del reloj.
  + CrearM (unsigned int): esta función tiene como parámetro un numero entero positivo, para crear y retornar un doble puntero a un arreglo bidimensional de enteros.
  + Validar (): esta función se encarga de pedir como entrada cada dato de la “regla” y comprobar si son válidos o no, por último, retorna un puntero a un arreglo unidimensional de enteros que sería la “regla”
  + comparar\_arreglos (int\*\*, int\*\*, unsigned int, unsigned int, unsigned int, unsigned int, int): esta función recibe como parámetros dos punteros dobles, ingresados en el orden que se encuentran uno detrás de otro, también se ingresan 4 enteros positivos equivalentes a las posiciones de los números que se van a comparar en los dos arreglos, y por ultimo se recibe un entero que equivale a la condición que deben cumplir los dos números a comparar. Esta función retorna una variable tipo bool dependiendo si se cumplen o no las condiciones dadas.
  + cambiar\_matriz (int\*\*&, int\*\*, unsigned int): esta función tiene como parámetros: la dirección de memoria de un doble puntero para reservar nueva memoria y crear un nuevo arreglo bidimensional “vacío” de las mismas dimensiones del segundo arreglo usando el siguiente doble puntero y el entero positivo que equivale a esa dimensión.
  + Liberar\_memoria (int\*\*, unsigned int): esta función tiene como parámetros un doble puntero a un arreglo bidimensional de enteros y un entero positivo que equivale a la dimensión de ese arreglo. Esta función toma el doble puntero y libera la memoria reservada para este.
* **Problemas en el desarrollo**

Durante el desarrollo del proyecto se presentaron diferentes tipos de errores a la hora de ejecutar el programa como los siguientes.

* Ingresar una Regla que tuviera tres 1 o más, por ejemplo K(4,3,1,1,1): este error implicaba que se mostrara un mensaje por pantalla de “Process crashed”, esto se debe a que para estos casos el programa rotaba el arreglo y siempre iba a tener de cerradura la dimensión del arreglo base, sin embargo llegaba un punto en que el numero en la posición dada ya no podía ser mayor a los números del arreglo base que se alinearan con este al rotar el arreglo, porque tendría que aumentar su dimensión pero al aumentar su dimensión solo encontrara número aún más grandes y al no poder crear un arreglo con dimensión más pequeña que la dimensión del arreglo base, el Código no tendrá fin y aparecerá el error “Process crashed”.
* Mal creación de un arreglo: cada vez se cumplía una condición se tenía previsto liberar la memoria del arreglo 1 y copiar todo el contenido del arreglo 2 porque este ultimo se tenia que actualizar de nuevo al arreglo base, sin embargo, hubo un problema con la función “cambiar\_matriz”, porque no estaba copiando los datos correctamente en el arreglo1. Este error se soluciono cambiando el primer parámetro de la función que pedía un doble puntero y en lugar de eso pedimos como parámetro la dirección de memoria de ese puntero para modificar los valores mediante esa dirección de memoria.
* Números incorrectos: aunque no es un error muy grave, si se paso por alto y es que al ingresar las posiciones del primer numero en la Regla, se tomo directamente estos valores para acceder al numero de cada arreglo, lo que origino que no se compararan los números correctos, pero la solución más simple fue restarle 1 a estos datos.
* Cambio de las posiciones dependiendo de la dimensión del arreglo: se tenía previsto que al terminar de rotar un arreglo y aun seguía sin cumplirse la condición, entonces se le aumentara +2 dimensiones al segundo arreglo y con esto asignábamos las direcciones del numero del arreglo 2 a las direcciones del numero del arreglo 1 y luego sumamos +1 a cada ubicación del arreglo 2, sin embargo eso solo era útil cuando el primer arreglo era menor al segundo, porque si el primer arreglo era mayor al segundo entonces se tomaría un numero en una posición errónea. Esto se soluciono aplicando un condicionar para estos dos casos descritos y cuando el arreglo 1 tuviera dimensión mayor a la dimensión del arreglo 2 entonces solo se le aumentara +1 a las ubicaciones del numero en el arreglo 2.
* **Evolución de la solución y consideraciones para tener en cuenta en la implementación**
  + Usar la función “validar” para pedir al usuario que ingrese los datos de K(regla), validando cada entrada.
  + Separamos la “fila” y la “columna” para guardarlos en dos arreglos (M, N) diferentes, uno para el arreglo1 y el otro para el arreglo2, además inicializamos una variable para llevar el control de cual condición es la que vamos a usar y a medida que se cumpla vamos aumentando el valor de esta variable para que pase a la siguiente condición.
  + Mediante condicionales y el uso de la “fila” y “columna” se determina la dimensión del arreglo base (B) y se tienen otras dos variables con esta misma dimensión que irán cambiando a lo largo del Código (C, D).
  + Se crean dos arreglos bidimensionales usando las dimensiones guardadas en C y D.
  + Mediante ciclos se compara el numero con la ubicación M del primer arreglo con el numero en la ubicación N del segundo arreglo. esta comparación se hará con la función “comparar\_arreglos” que me retornará “true” o “false” dependiendo si se cumple o no la condición.
  + Si la condición se cumple entonces se imprimirá la dimensión del arreglo 2 y se usara la función “liberar\_memoria” para el arreglo 1, luego se copiarán los datos del arreglo 2 al arreglo 1 usando la función “cambiar\_matriz” y luego se liberara la memoria del arreglo 2 para luego crear el arreglo 2 con las dimensiones del arreglo base. además, se deben actualizar el arreglo de posición del numero del primer arreglo (M), copiando los datos del segundo arreglo (N), luego se actualizará el arreglo N con la ubicación del número en el arreglo base, es decir, usando “fila” y “columna”.
  + Si la condición no se cumple entonces se rotará el arreglo 2 hasta que se cumpla la condición y se aplicará el paso anterior, pero si la condición no se cumple luego de rotar 3 veces el arreglo, entonces se liberará la memoria reservada para el arreglo 2 y se aumentará su dimensión +2 (D +2), luego se creará un nuevo arreglo usando el mismo doble puntero, pero con la nueva dimensión. Por último, si la dimensión del arreglo 1 es menor que la dimensión del arreglo 2, entonces se actualizara la ubicación del numero en el arreglo 1 y se colocaran los datos de la ubicación del numero en el arreglo 2 y a los datos de la ubicación del arreglo dos se le sumara +1. Si la dimensión del arreglo 1 es mayor a la dimensión del arreglo 2 solo se aumentará +1 a los datos de la ubicación del numero en el arreglo 2.
  + Por último, cada vez que se cumpla una condición entonces la variable de tipo entero que usamos para saber la condición se irá aumentando +1 y otra variable que contiene el numero de condiciones se ira restando -1. Como consecuencia cuando la variable que me guarde el numero de condiciones llegue a 0 se acabara el programa