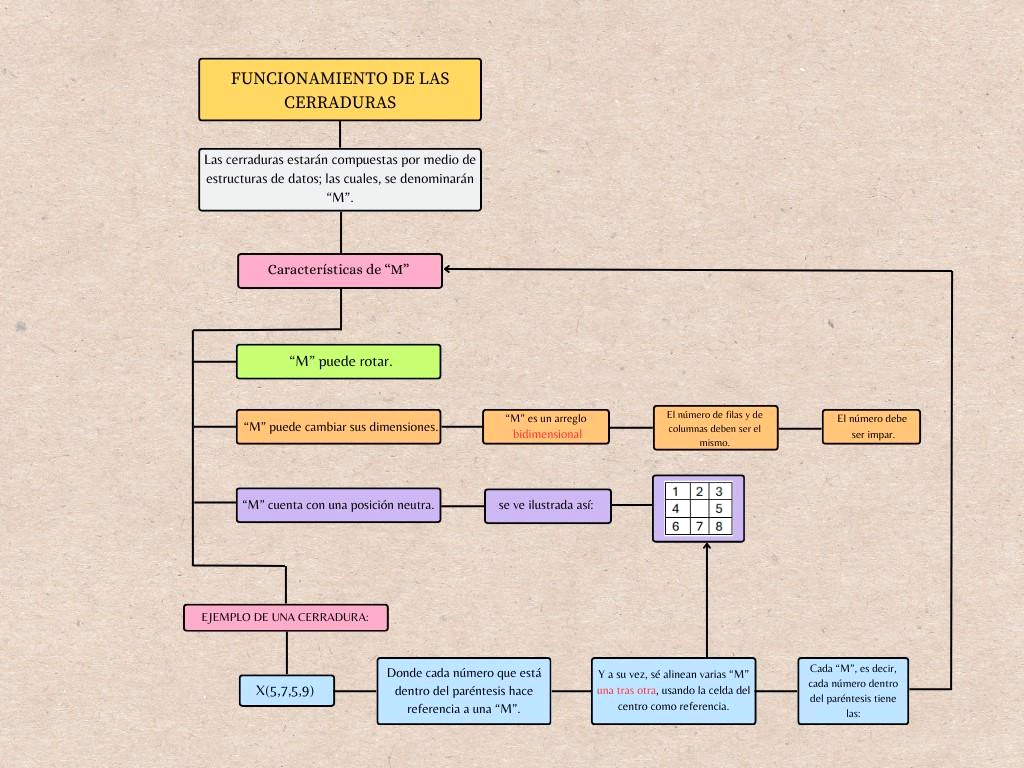
**Proceso de análisis y diseño de solución Parcial #1.**

El problema planteado en este desafío consiste en la validación de una clave que será la “llave” para abrir cerraduras; tanto claves como cerraduras tienen ciertas características específicas que deben ser cumplidas y validadas.

Se nos ha pedido diseñar un programa en C++ que sirva como mecanismo de solución frente a las necesidades que la empresa “Informa2” posee. Además, sé nos ha proporcionado una serie de ítems que el programa debe de cumplir para ser completamente útil y funcional de acuerdo a las especificaciones de la empresa.

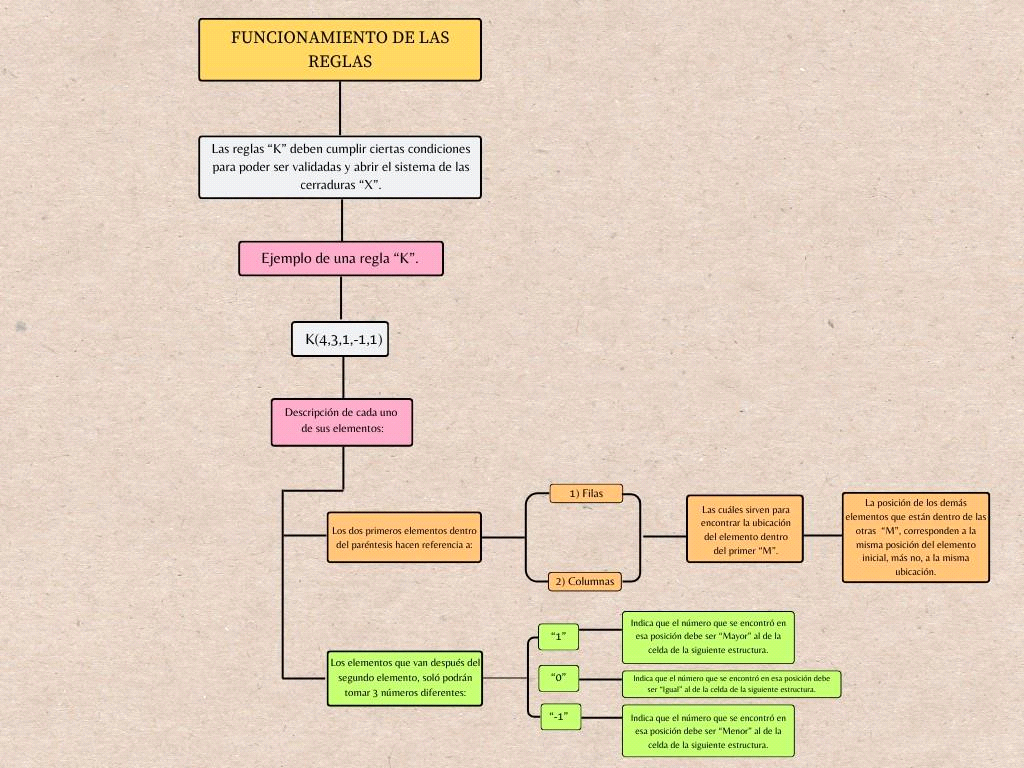
1. **Análisis del problema y consideraciones para la alternativa de solución propuesta.**



* **Análisis del problema:**

**Funcionamiento de las cerraduras fabricadas por la empresa “Informa2”:** A continuación, se planteará un esquema que contiene información referente a las características específicas de las cerraduras **“X”**; con el fin de tener una mejor legibilidad y organización sobre las características de las cerraduras

**Funcionamiento de las reglas “K”:**

A continuación, se planteará un esquema que contiene información referente a las características específicas de las reglas **“K”**; con el fin de tener una mejor legibilidad y organización sobre las características de las claves.

* **Consideraciones para la alternativa de solución propuesta:**

Teniendo en cuenta los dos esquemas anteriores y las características descritas sobre cada elemento **(cerradura y clave);** hemos diseñado un plan de desarrollo con el fin de resolver el problema planteado por la empresa “Informa2”.

Se pondrán las ideas a manera de ítems; en donde cada uno de ellos, corresponderá a la descripción de las estrategias que se utilizarán para resolver cada una de las tareas.

**Comportamiento de las cerraduras:**

* **“M” puede cambiar sus dimensiones:** Frente a esta característica en particular, sé creó una función llamada **“crear\_matrices”,** la cual está encargada de retornar un doble puntero a un arreglo bidimensional de **DIM** **variable**; es decir, que este arreglo hace uso de memoria dinámica, pues es una herramienta facilitadora qué permitirá ajustar el tamaño del arreglo según sea necesario, ya que este sólo sé conocerá durante el tiempo de compilación.
* **“M” puede rotar:** Para darle solución a esta característica, se crearon tres funciones llamadas: **“transponer”,** **“invertir\_filas” y “rotar\_noventa”, las cuales** funcionan en conjunto, pues la función **“transponer”** recibe un doble puntero a una matriz de DIM variable y la dimensión “n” de esa matriz como parámetros; luego, esta se transpone e invierte sus elementos en términos de columnas. Es decir, cada “i” (filas) de cada “M” va a tomar la posición de cada “j”(columnas) de cada “M”, después se recorren al revés las “i” de “M”, y de esta manera el arreglo “M” queda en una “**posición 1”** que es igual a 90°.
* **Matriz en posición neutral**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| IND | 0 | 1 | 2 | 3 | 4 |
| 0 | 1 | 2 | 3 | 4 | 5 |
| 1 | 6 | 7 | 8 | 9 | 10 |
| 2 | 11 | 12 |  | 13 | 14 |
| 3 | 15 | 16 | 17 | 18 | 19 |
| 4 | 20 | 21 | 22 | 23 | 24 |

* **Matriz en posición neutral transpuesta**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| IND | 0 | 1 | 2 | 3 | 4 |
| 0 | 1 | 6 | 11 | 15 | 20 |
| 1 | 2 | 7 | 12 | 16 | 21 |
| 2 | 3 | 8 |  | 17 | 22 |
| 3 | 4 | 9 | 13 | 18 | 23 |
| 4 | 5 | 10 | 14 | 19 | 24 |

* **Matriz en posición 1 (90°)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| IND | 0 | 1 | 2 | 3 | 4 |
| 0 | 5 | 10 | 14 | 19 | 24 |
| 1 | 4 | 9 | 13 | 18 | 23 |
| 2 | 3 | 8 |  | 17 | 22 |
| 3 | 2 | 7 | 12 | 16 | 21 |
| 4 | 1 | 6 | 11 | 15 | 20 |

El numeral “B” y “C” se repiten dos veces más, permitiendo que se creen dos nuevas posiciones para el arreglo, las cuales corresponden a 180° y 270° respectivamente.

* **“M” cuenta con una posición neutra:** Tal cómo se describió en el esquema anterior, cada “M” que haga parte de la cerradura “X” contará con una posición neutra, la cual servirá como referencia para que cada una de las “M” se alineen una detrás de la otra, usando la celda del centro como referencia.

De acuerdo a lo anterior, dentro de la función **“crear\_matrices”** sé implementó una lógica adecuada para hacer esto posible, usando “for” anidados que iteren sobre cada elemento del arreglo bidimensional, de tal manera que en el centro de este, se encuentre una posición neutra.

**Ejemplo:** Tengo una estructura "M" de 5 filas y 5 columnas, y si tomamos en cuenta lo anterior, la posición neutra debería ser la celda que está ubicada en la fila 2 y la columna 2.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| IND | 0 | 1 | 2 | 3 | 4 |
| 0 | 1 | 2 | 3 | 4 | 5 |
| 1 | 6 | 7 | 8 | 9 | 10 |
| 2 | 11 | 12 |  | 13 | 14 |
| 3 | 15 | 16 | 17 | 18 | 19 |
| 4 | 20 | 21 | 22 | 23 | 24 |

De acuerdo a lo anterior, la celda encontrada en cada “M” sería la posición neutra, pues así lo plantean las características del problema que estamos solucionando.

**Comportamiento de las claves:**

**Análisis de los elementos que componen a “K”:**



De acuerdo a la imagen anterior, podemos evidenciar una serie de elementos que componen la clave, la ilustración anterior permite tener una mejor comprensión sobre la cantidad de elementos que componen a “K”.

1. En primera instancia sé creó una función llamada **“ingresar\_numeros”**; la cual se encarga de pedirle al usuario los elementos de la clave “K” y de guardarlos en un arreglo dinámico.

**EN EL MAIN:**

1. En el **“main”,** se empieza a manipular la dirección de memoria que retorna la función “ingresar\_numeros” para poder tener acceso a los diferentes elementos que contiene la clave “K”; es importante resaltar que el “arreglo\_k” contendrá los elementos de “K”, y también el número “-100” ( -100, es el número que el usuario debe ingresar por pantalla para indicar que ya no ingresará más números)
2. Los valores de los 2 primeros elementos del “arreglo\_k”, correspondientes a la “fila” y la “columna” se obtendrán por medio de indexación, en la posición “0” y “1”, y se guardarán en la variable fila1 y columna1 respectivamente.
3. **Dentro del “main” sé construirá un “while” para obtener dos variables:**

- La primera se llamará **“dato”,** y esta irá almacenando cada elemento del “arreglo\_k” mientras la condición del while se cumpla.

- La segunda se llama **“contador”,** y esta almacenará el # de elementos del “arreglo\_k” (incluyendo el -100).

1. Sé definirá una variable llamada **“Tamanio\_K”** la cuál contiene el # de elementos de K, sin incluir el -100; es decir, la DIM del “arreglo\_k”.
2. Sé definirá una variable llamada **“cantidad\_condiciones”**, la cual almacenará la cantidad de condiciones que tiene el “arreglo\_k”. La cantidad de condiciones se obtiene restándole “2” al “Tamanio\_k”. (“2” corresponde a los dos primeros datos del “arreglo\_k”, los cuáles son, filas y columnas.)
3. **Posterior a esto, se crearán una serie de condicionales, que harán comparaciones con los siguientes casos:**

- si “fila1” es mayor que “columna1”: sé verificará si “fila1” es par; en caso de serlo, se le suma “1” a la variable “fila1”; de lo contrario “fila1” conserva su valor inicial.

- si “columna1” es mayor que “fila1”: sé verificará si “columna1” es par; en caso de serlo, se le suma “1” a la variable “columna1”; de lo contrario “columna1” conserva su valor inicial.

-Si “fila1” es igual a “columna1”: sé verificará si “fila1” es par; en caso de serlo, se le suma “1” a la variable “fila1”; de lo contrario “fila” conserva su valor inicial.

Todas las condiciones anteriores se planean hacer con la finalidad de encontrar la dimensión **inicial** de los arreglos de la cerradura “X”, la cual se almacenará en la variable “dim\_M1”.

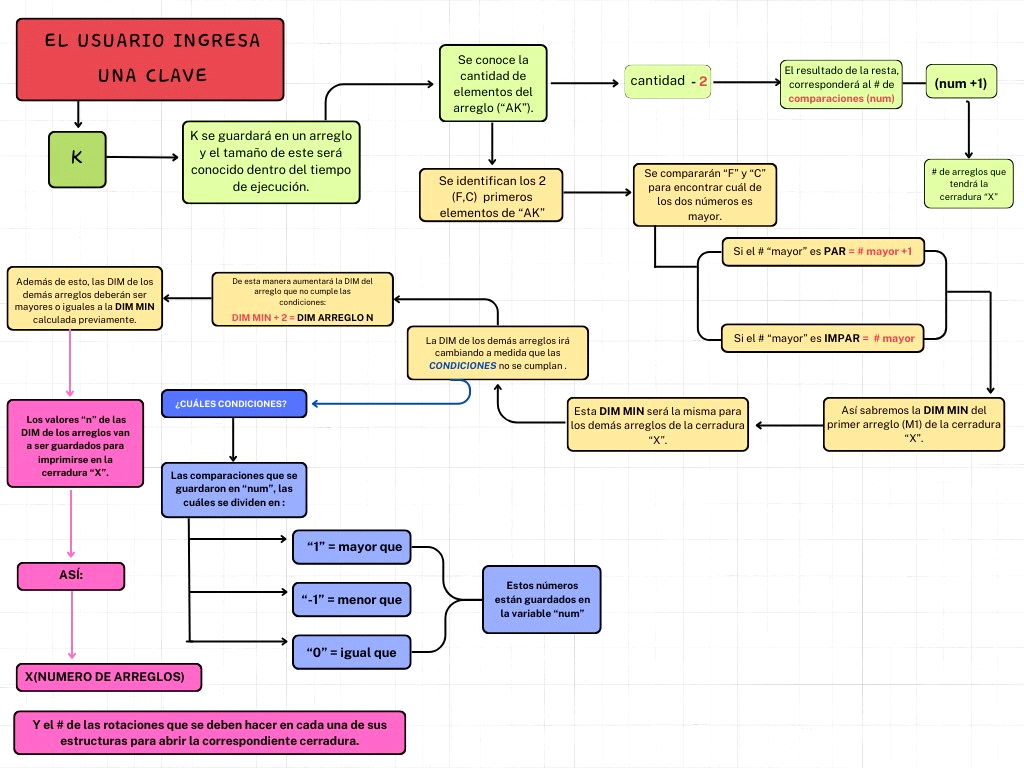
1. Se crearán tres variables llamadas “fila2”, “columna2” y “dim\_variable”; las cuales tendrán el mismo valor que las variables “fila1”, “columna1” y “dim\_M1” respectivamente. Con el fin de tener una coordenada para encontrar un elemento determinado en la Matriz2 que está detrás de la Matriz1, y así sucesivamente con la cantidad de arreglos que tenga mi cerradura X.
2. Posteriormente, se creará un arreglo bidimensional con la DIM almacenada en la variable “dim\_M1” y se guardará en un puntero doble; es decir, que en este punto se está creando el primer arreglo de la cerradura “X”.
3. Sé definirá una variable llamada “dim\_variable1”, la cual tendrá el mismo valor que la variable “dim\_M1” con el fin de que la DIM de la nueva matriz sea igual a la primera.
4. Para saber el número de arreglos de mi cerradura “X”, simplemente se le sumará “1” a la variable “cantidad\_condiciones”, porque de acuerdo a la lógica utilizada en el programa, las matrices se comparan de a dos en dos.Y ese dato se guardará en una variable llamada “cantidad\_de\_ArrdeX”.
5. Posteriormente se creará un arreglo de enteros llamado “arreglo\_X” con la dimensión=”cantidad\_de\_ArrdeX”; es decir, que en “arreglo\_X” se guardarán los números que hacen referencia a las dimensiones de las matrices que compondrán la cerradura “X”.
6. También se creará un arreglo de enteros llamado “arreglo\_rotaciones”, el cuál tendrá la misma dimensión que el “arreglo\_X”, pues en este se almacenarán la cantidad de veces que tuvo que rotar cada matriz de la cerradura “X” respectivamente.

**Ejemplo:**

arreglo\_X = X (5, 7, 5, 5, 5)

arreglo\_rotaciones = R(0, 0, 0, 0, 1)

Según el ejemplo anterior, la matriz de 5x5 rotó 0 veces, la matriz de 7x7 rotó 0 veces, y así sucesivamente.

1. Posteriormente, se creará un arreglo bidimensional con la DIM almacenada en la variable “dim\_variable” y se guardará en un puntero doble; es decir, que en este punto se está creando el segundo arreglo de la cerradura “X”.
2. En este punto, se tienen dos arreglos que se pasarán como parámetros a la función “comprobador”, esta devolverá un booleano TRUE o FALSE, que permitirán determinar si se debe rotar la matriz ó si se debe aumentar de DIM.
3. Con lo anterior se tendrán las rotaciones necesarias para los arreglos que pertenecen a la cerradura “X” respectivamente.
4. Y de esta manera, se obtienen los datos que se deben imprimir en pantalla.
5. **Esquema donde describa las tareas que usted definió en el desarrollo de los algoritmos.**
6. **Algoritmos implementados.**

**Funciones implementadas en el programa**

**Función crear\_matrices=** Esta función crea una matriz bidimensional cuadrada de tamaño nxn. Cada elemento de la matriz se asigna con un valor que incrementa secuencialmente desde 1 hasta nxn, excepto el elemento en el centro de la matriz, el cual es igual a 0.

**Función ingresar\_numeros=** Esta función devuelve un puntero a un arreglo de tamaño variable, el cuál contiene los elementos de la clave “K”, ingresado por el usuario.

**Función comprobador=** Esta función verifica si una condición específica entre dos elementos de dos matrices es verdadera o no. La función toma como parámetros dos matrices bidimensionales (matriz\_1 y matriz\_2), junto con las coordenadas de dos elementos en esas matrices (fila1, columna1, fila2, columna2) y una condición.

**Función cambiar\_matriz=** Esta matriz copia los valores de una matriz (matriz2) en otra matriz (matriz1) del mismo tamaño. La función toma como parametros dos matrices bidimensionales (matriz1 y matriz2) y un entero (dimension) que especifica la dimensión de ambas matrices.

**Función liberar\_mem\_neutra**= Esta función se encarga de liberar el espacio de memoria asignado dinámicamente para una matriz bidimensional.

**Función transponer=** Esta función realiza la operación de transposición en una matriz cuadrada de tamaño nxn. La transposición implica cambiar las filas de una matriz por sus columnas correspondientes.

**Función invertir\_filas=** Esta función invierte las filas de una matriz bidimensional de tamaño nxn. La función itera solo sobre la mitad de las filas de la matriz y, para cada fila, intercambia sus elementos con los elementos correspondientes de la fila simétrica en relación con el eje horizontal que pasa por el centro de la matriz.

**Función rotar\_noventa=** Esta función rota la matriz en sentido antihorario 90 grados. Para lograr esto, utiliza dos operaciones: la transposición y la inversión de filas.

**D ) Problemas de desarrollo que afrontamos:**

* Entender una estructura multidimensional, y plantear el funcionamiento para aprender a recorrer dichos arreglos.
* Comprender el uso de la memoria dinámica y cómo emplearlo en el programa, y el cómo crear una estructura bidimensional de tamaño variable.
* Tener en cuenta la utilidad y la sintaxis de los punteros.
* La utilización oportuna del delete[].

E) **Evolución de la solución y consideraciones para tener en cuenta en la implementación.**

Partir del diseño de un informe inicial se convierte en un ejercicio positivo y negativo a la vez, pues el determinar una ruta es un avance para el inicio, pero reconocer que no todo lo planteado funciona y que requiere de cambios significativos cuando se empieza a implementar lo propuesto, juega significativamente en todo el proceso, donde el tiempo es una de las variables más importantes a tener en cuenta, pues se debe contemplar acciones no solo de diseño de códigos si no también acciones de análisis, comprensión, interpretación creatividad, ensayo y error que fueron cambiando en la medida en que se iba reconociendo cuales serían las rutas mas largas o cortas para hacer mucho mas optimo el código.

La evolución de la solución empieza a coger forma cuando se lleva lo que se piensa al plano de la grafica o del dibujo, facilitando el análisis y la interpretación de lo que se debía trabajar.

Es importante tener en cuenta al momento de implementar el código, el manejo de punteros, memoria dinámica, el tipo de variables y la capacidad que estas ocupan en la memoria para garantizar un óptimo uso de los diferentes recursos que se requieren y la entrega de una solución efectiva pero sencilla en su desarrollo.