Informe Desafío 1 – Juan Pablo Avendaño y Freddy Alexander Castaño.

Análisis del problema.

El desafío pide que se realice un algoritmo capaz de entregar al menos una configuración de candado que cumpla con la regla entregada. Además de eso en cada configuración va a haber solo matrices cuadradas impares y el ángulo de rotación entre un estado y el siguiente va a ser de 90°.

Teniendo muy presente las consideraciones iniciales nos resultó fácil idear una solución a este problema, primero que todo notamos que rotar las matrices completamente con el fin de pasar de un estado a otro era totalmente innecesario, lo que se hizo fue desarrollar una función que devolviera una celda dada según el estado en que se encontrara la matriz, es decir, si se ingresa como celda deseada '[4,1]' , es decir fila 4 y columna 1, y como estado ponemos el 3, la función devolverá el número que se encontraría en la celda dada si se hubiera rotado la matriz hasta el estado 3. Como la función que permitía rotar la matriz completamente era un requisito entonces decidimos implementar esta función con fines netamente de experiencia de usuario, es decir, solamente se implementó este requisito para mostrarle al usuario la matriz.

Continuando con la solución fue evidente para nosotros que en la regla iba a estar el tamaño mínimo de cada una de las matrices que fueran a componer el candado, por lo tanto si en la regla para abrir el candado se especificaba la celda '[4,3]', se tomaría el número mayor 'n' entre las dos posiciones de la celda y en el caso de que fuera par, entonces se crearía una matriz de orden n+1, si el número es impar, entonces se crearía una matriz de orden n, si la celda corresponde a la posición '[1,1]', entonces se creará una matriz de orden 3, también se hace la verificación de que la celda no corresponda al centro de la matriz.

En cuanto a la validación de la regla se refiere definimos que ésta será invalida si después de la posición de la celda deseada se encuentran números diferentes a '1', '0' y '-1', o bien si directamente no existen números. La cantidad de matrices que se va a crear para la configuración del candado será la cantidad de elementos almacenados en la estructura de datos donde se guardará la regla (en este caso usamos un arreglo lineal de enteros alojado en el heap).

A continuación, consideramos pertinente crear una estructura que nos permitiera almacenar las matrices creadas, la utilidad de este componente la verá más adelante.

En cuanto a generación de la configuración se refiere nosotros consideramos un algoritmo que se encargara de encontrar una combinación válida entre dos matrices que cumpliera una regla dada (por regla nos referimos a que una celda sea mayor, menor o igual a otra). La rotación de ambas matrices hasta encontrar una combinación valida es una de las características de este algoritmo, si no se encuentra tal combinación simplemente retornará un valor false, si la combinación es válida entonces se retornará true y además de eso se editará un arreglo de enteros donde la posición 0 indica el estado de la matriz uno, y la posicion 1 indica el estado de la matriz dos, cabe recalcar que el algoritmo permite dejar una o ambas matrices estáticas, es decir que no van a rotar, esto es útil para la generación completa de la configuración.

Para concluir la solución del problema notamos que la primera comparación entre matrices que fueran a componer mi candado podría tener cualquier orden y estado de rotación, pero las siguientes comparaciones deberían tener una matriz estática, es decir, con estado invariable, y

este estado sería el estado de la segunda matriz a comparar en la comparación anterior, a continuación, se da un ejemplo para que visualice mejor el concepto:

Tenemos una estructura de 4 matrices con una regla general dada, para la primera combinación se obtuvo un resultado donde la primera matriz debía tener el estado 2 y la segunda el estado neutral para cumplir el primer criterio de la regla. Seguido a eso debe comparar la segunda matriz con la tercera, pero en este caso la segunda matriz debe conservar el mismo estado que se obtuvo en la primera comparación, y así mismo pasa con la tercera y cuarta comparación. Si este aspecto no es tomado en cuenta el resultado final será con toda seguridad erróneo.

Finalmente, se recorre toda la estructura de matrices haciendo las comparaciones necesarias entre pares de matrices, cuando se finalice la comparación entre dos matrices, la segunda de ellas se pondrá estática, es decir no se alterará su orden o estado de rotación, esto con el fin de que las matrices sigan cumpliendo los criterios de comparación que se cumplieron en las comparaciones anteriores.

Finalmente se creará una matriz donde se van a almacenar los resultados obtenidos, la matriz resultado tendrá ‘n’ filas, donde n es el número de matrices en la configuración del candado, y la matriz tendrá 2 columnas. En la primera columna se ubica el orden de la matriz (matriz que compone el candado, no la matriz resultado), y en la segunda irá el estado de rotación de la misma.

Y, por último, se le despliega al usuario los resultados que se obtuvieron de manera que pueda visualizar cada matriz.

Esquema.

A continuación, se presentará un esquema de los dos módulos principales de la solución en donde se resuelve el problema, estos son: **Estructura** y **Candado**.

Protocolo: Estructura

Función crearMatriz(Se guarda la matriz en la memoria que retornara que retornara)

Se iteran las filas

Se itera las columnas dentro del argumento de las filas.

Se posiciona el espacio en blanco, si cumple la condición.

Si cumple con la división tanto la fila como la columna.

Se posicionan los números consecutivos en la matriz

Cada suma del contador, es un numero en tal posición.

Función crearEstructura(Se guarda en la memoria el arreglo de matrices que retornara.)

La cantidad de matrices es el tope de la iteración.

Por cada iteración se crea una matriz de x orden, y se guarda en el arreglo dinámico.

Función obetnerRotacion(Retornara la celda que esta en la matriz una vez halla sido o no rotada)

Conociendo la mitad, y las posiciones de las celdas elegidas por el usuario, iniciamos la rotación por medio de una formula que se le asigna a la columna

Las otras rotaciones se harán con la repetición de la primera. En caso de no haber rotación, solo se asignara la columna y la fila original.

La columna se le asigna a la fila, y la formula que es 2 veces la mitad restando con la fila original(sin ser modificada la columna): Da la rotación de 90 grados.

Función generarOrdenes(Crea un arreglo de números que representan las matrices con la misma dimensión, para así ser usado en en la función generarCandado, que esta en otro protocolo para crear las matrices de la combinación K)

El parametro tamano, nos indica el numero de posibles matrices que se van a comparar

Una vez iterado, con la variable de control tamano, se crea un arreglo de un numero en diferentes posiciones, y los retorna.

Función editarEstructura(Elimina y agrega otra matriz)

Se libera de la memoria la matriz que no encaja con la combinación, al comparar las dos celdas de las dos matrices.

En ese misma posición de comparación de las dos matrices, se agrega la nueva matriz.

Función destruirEstructura(Libera la memoria de la n matrices que fueron creadas)

Protocolo: Candado.

Función validarRegla(Nos indica si todas la condiciones para evaluar el candado se cumplen)

Valida si las posiciones 0 y 1 son mayores que 0, porque no hay columnas y filas negativas.

Valida si el arreglo es mayor que 2. Pues en caso de que no, no existirían matrices para comparar.

Evalua si cada posicion de las matrices cumple con los criterios de 1,-1 y 0.

Función encontrarComb(Retorna si de verdad las dos matrices que se compararon coinciden con las configuraciones 1,-1 o 0)

Se creo un nuevo parámetro para construir la matriz.se llama regla (fila, columna, criterio(0,1,-1), falso o verdadero (0 o 1), falso o verdadero)

Los falsos y verdaderos, determinaran si la matriz roto o no.

Si la matriz es estatica se define cada celda a comparar

Si se itera, es porque no es estática, y debe hallarse una que lo sea.

Luego se halla el criterio que coincida con las celdas que se comparan.

Si no cumple con los criterios retornara falso.

Se guarda en un arreglo, las veces que rotaron las matrices en cada posición. Siempre y cuando cumplan con los criteiros.

Función generarCandado(Es construir con ayuda de todas la funciones creadas, las celdas correctas, que a su vez dependa de la matriz correcta, en la rotación correcta)

Inicia con una iteración del arreglo resultado, para ubicar ahí mas adelante el orden de la matriz y su rotación.

Se construye la matriz que coincida con la fila y columna de la regla, y ese será el orden mínimo.

Se comprueba si la combinación es verdadera, si no lo es, se edita la estructura de la matriz

Si es verdadera la combinación de la matriz, se le asigna la dimensión y el estado al resultado. Que es lo que retornara.

Ambas funcionan con dos iteraciones: un for para asignar las matrices a comparar, y un while anidado en el for para buscar la combinacion, editar las matrices (sea el caso) y ubicar los estados de rotacion.

Algoritmos implementados.

En el desarrollo del código implementamos los dos módulos principales de los cuales fueron explicadas las tareas más representativas como pudo ver en el esquema anterior, en esta sección ahondaremos un poco más en la implementación de todos los módulos, repasando cada una de las funciones que lo componen:

Módulo Estructuras:

- **Función crearMatriz:**

```
/**
 * Breve descripción de la función crearMatriz.
 *
 * Permite crear una matriz y llenarla segun las consideraciones iniciales.
 *
 * @param orden Orden de la matriz deseada.
 * @return Devuelve un puntero que apunta al primer elemento de la matriz.
 */

int** crearMatriz(int orden){
    int medio = (orden / 2); //Se encuentra el punto medio de la matriz.
    int contAux = 1; //Se inicia un contador auxiliar que va a ser el primer numero que se ubique en la posicion [0][0] de la matriz.
    int **matriz = new int*[orden]; //Se aloja memoria para la matriz.
    for(int fila = 0; fila < orden; fila++){
        matriz[fila] = new int[orden]; //Se crea un arrelgo de enteros en cada fila de la matriz.
        for(int columna = 0; columna < orden; columna++){
            if(fila == medio && columna == medio){ //Si se trata de la coordenada media de la matriz, entonces se asigna -1 y no se actualiza el contador auxiliar.
                matriz[fila][columna] = -1; //El -1 indicará que es un espacio en blanco mas adelante.
            }
            else{
                matriz[fila][columna] = contAux; //Se asigna el valor del contador auxiliar si la coordenada actual no es el punto medio.
                contAux++;
            }
        }
    }
    return matriz;
}
```

```
int** crearMatriz(int orden){
    int medio = (orden / 2);
    int contAux = 1;
    int **matriz = new int*[orden];
    for(int fila = 0; fila < orden; fila++){
        matriz[fila] = new int[orden];
        for(int columna = 0; columna < orden; columna++){
            if(fila == medio && columna == medio){
                matriz[fila][columna] = -1;
            }
            else{
                matriz[fila][columna] = contAux;
                contAux++;
            }
        }
    }
    return matriz;
}
```

Esta función permite crear la estructura cuadrada que se menciona en las consideraciones iniciales del problema, recibe solamente el orden de la estructura que se desee crear, incluye dos ciclos ideados para recorrer completamente la estructura y llenarla de números, si la posicion corresponde a la del medio se le asigna el numero -1, que después será interpretado como un espacio en blanco.

- **Función crearEstructura:**

```
/**
 * Breve descripción de la función crearEstructura.
 *
 * Permite crear un arreglo de matrices.
 *
 * @param ordenes Arreglo con todos los ordenes de las matrices deseadas.
 * @param numMatrices Cantidad de matrices que habra en la estructura.
 * @return Devuelve un puntero que apunta al primer elemento de la primera matriz de la estructura.
 */

int ***crearEstructura(int *ordenes, int numMatrices){
    int ***estructuraPtr = new int**[numMatrices]; //Se aloja memoria para la estructura
    for(int cont = 0; cont < numMatrices; cont++){ //Se llena la estructura haciendo uso de la funcion crearMatriz.
        estructuraPtr[cont] = crearMatriz(ordenes[cont]);
    }
    return estructuraPtr;
}
```

Esta función permite crear un arreglo de matrices al cual llamaremos estructura, en el codigo se utiliza esta estructura para poder tener acceso a cualquier celda de las matrices que compongan vayan a componer mi configuración de candado, se entrega mediante un puntero la posicion de memoria de esta estructura (el primer elemento de la primera matriz).

- **Función obtenerRotacion:**

```
/**
 * Breve descripción de la función obtenerRotacion.
 *
 * Permite obtener el numero en una celda dada de una matriz que tiene un estado dado.
 *
 * @param posActual Arreglo con las coordenadsa de la celda deseada.
 * @param matriz Matriz a la cual se le quiere obtener la celda (debe estar en su estado neutral).
 * @param orden Orden de la matriz.
 * @param estado Estado de la matriz.
 * @return Devuelve un entero que es el valor que tendria la matriz en la celda dada si estuviera en un estado dado.
 */

int obtenerRotacion(int *posActual, int **matriz, int orden, int estado){
    int medio = orden / 2; //Ubica el punto medio de la matriz.
    int fila = posActual[0]; //Se define la fila y columna, creando a su vez una copia de esta fila para no perder el valor
    int filaAux = fila;
    int columna = posActual[1];
    fila = columna; //Se identifica que cuando se va de un estado a otro la fila se vuelve la columna.
    columna = 2*medio - filaAux; //Se identifica tambien que la columna va a ser igual a la diferencia entre el doble del medio y la fila dada.

    if(estado == 1) return matriz[fila][columna]; //Se devuelve el valor requerido si el estado es 1, sino sucede esto entonces se realiza el mismo proceso hasta el estado 3.
    else if(estado == 2){
        filaAux = fila;
        fila = columna;
        columna = 2*medio - filaAux;
        return matriz[fila][columna];
    }
    else if(estado == 3){
        filaAux = fila;
        fila = columna;
        columna = 2*medio - filaAux;
        return matriz[fila][columna];
    }
    else return matriz[posActual[0]][posActual[1]]; //Si el estado es 0, entonces se devuelve el valor de la matriz en la fila y columna originales.
}
```

Esta funcion permite simular la rotación de una matriz y obtener el número que estaría en la celda especificada si el estado de la matriz fuera el que se ingresó en la funcion, nos dimos cuenta de que rotar la matriz completamente para cada comparación era un esfuerzo computacional innecesario, por lo tanto, a base de tanteo dedujimos una relacion para hallar aquel valor de la matriz en la celda especificada con el estado especificado.

- **Función generarOrdenes:**

```
/**
 * Breve descripción de la función generarOrdenes.
 *
 * Genera un arreglo los cuales todos tienen el mismo valor, que va a ser un orden dado.
 *
 * @param orden Valor que deben compartir todos los elementos del arreglo.
 * @param tamano Tamaño del arreglo.
 * @return Devuelve el puntero que esta asociado al primer elemento del arreglo.
 */

int *generarOrdenes(int orden, int tamano){
    int *ordenes = new int[tamano]; //Se crea un arreglo de enteros.
    for(int cont = 0; cont < tamano; cont++) ordenes[cont] = orden; //Para cada celda se le asigna el mismo valor.
    return ordenes;
}
```

Esta funcion simplemente genera un arreglo de números que tiene un número fijo en cada elemento de él, en este caso el orden especificado, será de mucha utilidad en otras funciones ya que permitirá obtener los órdenes de las matrices que intervienen en la generación del candado, y además de eso se podrá editar este orden a conveniencia para encontrar una combinación.

- **Función editarEstructura:**

```
/**
 * Breve descripción de la función editarEstructura.
 *
 * Permite obtener el numero en una celda dada de una matriz que tiene un estado dado.
 *
 * @param indice Posicion del arreglo de matrices (estructura) que va a ser editado.
 * @param ordenNueva Orden de la nueva matriz que va a ser insertada en la posicion de la vieja matriz.
 * @param estructura Arreglo de matrices (estructura) al cual se le quiere hacer el cambio.
 * @return Ninguno, simplemente edita por referencia la estrucutra.
 */

void editarEstructura(int indice, int ordenNueva, int ***estructura){
    delete [] estructura[indice]; //Se libera la memoria de la vieja matriz.
    estructura[indice] = crearMatriz(ordenNueva); //Se inserta la nueva matriz a la estructura
}
```

La funcion editarEstructura tiene como utilidad principal borrar completamente una matriz de una estructura de matrices, y seguido a eso se crea una nueva de un orden especificado, es útil cuando se estan comparando dos matrices y una de ellas impide que el criterio de comparación se cumpla, entonces se aumenta el orden de la matriz de una de ellas (se debe borrar la matriz anterior e insertar una nueva) y se compara otra vez hasta encontrar la combinación valida.

- **Función formarRegla:**

```
/**
 * Breve descripción de la función formarRegla.
 *
 * Genera un arreglo que contiene la regla ingresada por el usuario.
 *
 * @param fila Arreglo con las coordenadsa de la celda deseada.
 * @param columna Matriz a la cual se le quiere obtener la celda (debe estar en su estado neutral).
 * @param criterios Orden de la matriz.
 * @return Devuelve un arreglo que contine la regla ingresada por el usuario.
 */

int *formarRegla(int fila, int columna, string criterios){
    int *regla = new int[2 + criterios.size()]; //Se aloja memoria para el arreglo deseado, en este caso
    regla[0] = fila - 1; //Se le resta 1 a la celda y columan ingresada por el usuario.
    regla[1] = columna - 1;
    for(int cont = 0; cont < criterios.size(); cont++){ //Recorre todos los elementos del string criteric
        if(criterios.at(cont) == '1') regla[2 + cont] = 1;
        else if(criterios.at(cont) == '2') regla[2 +cont] = 0;
        else regla[2 + cont] = -1;
    }
    return regla;
```

La funcion formarRegla retorna el arreglo que se le asigno al puntero regla, que contiene los criterios de rotación de cada una de las matrices y la celda a evaluar. Esto tiene en cuenta el ingreso del menú (La opción 1 que está relacionada con la regla 1, la opción 2 que está relacionada con la regla 0, y la opción 3 que está relacionada con la regla -1), para definir que combinación desea el usuario. Entonces en los criterios se disponen a partir de la posicion 2 del arreglo.

Modulo Candado.

- **Función validarRegla:**

```
/**
 * Breve descripción de la función validarRegla.
 *
 * Permite validar una regla ingresada por el usuario
 *
 * @param regla Puntero que debe ser un arreglo de enteros, y representa la regla del candado.
 * @param sizeRegla Entero que representa el tamaño de la regla.
 * @return Devuelve un booleano queq indica si la regla es valida o no.
 */

bool validarRegla(int *regla, int sizeRegla){
    int ordenMin; //Se inicializa el orden minimo que va a tener un elemento de
    int medio; //Se inicializa el valor medio de la matriz de orden minimo.
    if(regla[0] < 0 || regla[1] < 0) return false; // Verifica que las coordenadas de la celda no sean negativas
    if(sizeRegla <= 2) return false; //Verifica que la longitud de la regla no sea inferior a 3
    for(int cont = 2; cont < sizeRegla; cont++){
        if(regla[cont] != 0 && regla[cont] != -1 && regla[cont] != 1) return false; //Para cada elemento de la regla verifica si es -1, 1 o 0.
    }

    //Las lineas de la 25 - 35 verifican que la celda no tenga coordenadas del punto medio de la matriz.
    if(regla[0] >= regla[1]){
        ordenMin = regla[0];
    }
    else ordenMin = regla[1];

    if(ordenMin % 2 == 0) ordenMin += 1;
    else ordenMin += 2;

    if(regla[0] == regla[1] && regla[0] == 0) ordenMin = 3;
    medio = ordenMin /2;
    if(regla[0] == medio && regla[1] == medio) return false;

    return true; //Entrega el boolean true, que indica que la regla es valida.
```

La funcion validarRegla consiste en verificar si lo ingresado por el usuario cumple con la regla. Las reglas que se deben cumplir son: que las filas y columnas sean enteros positivos, que realmente se ingresen los criterios adecuados, y que la celda no corresponda a un espacio en blanco en el centro de la matriz.

- **Función encontrarComb:**

```
bool encontrarComb(int **matrizUno, int **matrizDos, int *regla, int*arreglo, int* ordenes, int estadoUno, int estadoDos){
    int celdaUno; //Define variables para el numero que hay en la celda a comparar en la matriz uno y dos.
    int celdaDos;
    bool isEstaticaUno = regla[3]; //Define booleanos que indican si la matriz uno o dos son estaticas.
    bool isEstaticaDos = regla[4];
    int criterio = regla[2]; //Criterio de comparacion entre las dos matrices (1, -1 ,0).
    int *celda = new int[2]; //Arreglo que contendra las coordenadas de la celda.
    bool finalizado = false; //Indica si se encontro la combinacion deseada.
    celda[0] = regla[0]; //Asigna a las posicoines del arreglo celda, las coordenadas contenidas en la regla.
    celda[1] = regla[1];


    //De la linea 157 a 162 se le asigna una celda invariable a la matriz correspondiente en el caso de que sea estatica.
    if(isEstaticaUno){
        celdaUno = obtenerRotacion(celda, matrizUno, ordenes[0], estadoUno);
    }
    if(isEstaticaDos){
        celdaDos = obtenerRotacion(celda, matrizDos, ordenes[1], estadoDos);
    }

    for(int cont = 0; cont < 4; cont++){ //Ciclo while que va a iterar entre todos los estados posibles de la matrizUno, del 0 al 3.
        if(!isEstaticaUno) { //Se obtiene el valor de la celda dada en el estado que indique el contador.
            celdaUno = obtenerRotacion(celda, matrizUno, ordenes[0], cont);
        }
        for(int cont2 = 0; cont2 < 4; cont2++){ //Ciclo while que va a iterar entre todos los estados posibles de la matrizDos, del 0 al 3.
            if(!isEstaticaDos){ //Se obtiene el valor de la celda dada en el estado que indique el contador2.
                celdaDos = obtenerRotacion(celda, matrizDos, ordenes[1], cont2);
            }

            //En 173 - 182 se evalua si los numeros en las celdas de las dos matrices cumplen cada uno de los criterios.
            if(criterio == -1 && celdaUno < celdaDos){
                finalizado = true;
            }
            else if(criterio == 1 && celdaUno > celdaDos){
                finalizado = true;
            }
            else if(criterio == 0 && celdaUno == celdaDos){
                finalizado = true;
            }

            if(finalizado){ //Si las celdas cumplen el criterio se modifica el arreglo 'arreglo' y se le pone el estado para cada matriz que permitio que el criterio se cumpliera.
```

```
        if(finalizado){ //Si las celdas cumplen el criterio se modifica el arreglo 'arreglo' y se le pone el estado para cada matriz que permitio que el criterio se cumpliera.  You, 2024-04
            if(isEstaticaUno) arreglo[0] = estadoUno;
            else arreglo[0] = cont;

            if(isEstaticaDos) arreglo[1] = estadoDos;
            else arreglo[1] = cont2;

            delete [] celda; //se libera memoria
            return true;
        }
        if(isEstaticaDos){ //Si es estatica se lleva el contador a 3, esto con el fin de terminar el ciclo.
            cont2 = 3;
        }
    }
    if(isEstaticaUno){ //Si es estatica se lleva el contador a 3, esto con el fin de terminar el ciclo.
        cont = 3;
    }
}

delete [] celda; //Se libera memoria
return false;
}
```

La funcion encontarComb ayuda a encontrar la combinación de estados entre dos matrices que valide un criterio a evaluar, esta funcion da la posibilidad de dejar las matrices estáticas, es decir, su estado se mantendrá como el establecido como predeterminado, si encuentra la combinación que valide el criterio entonces se devolverá un booleano true, y a su vez se modificará por referencia un arreglo, indicando los estados que validaron la combinación, en el caso contrario se devolverá un booleano false.

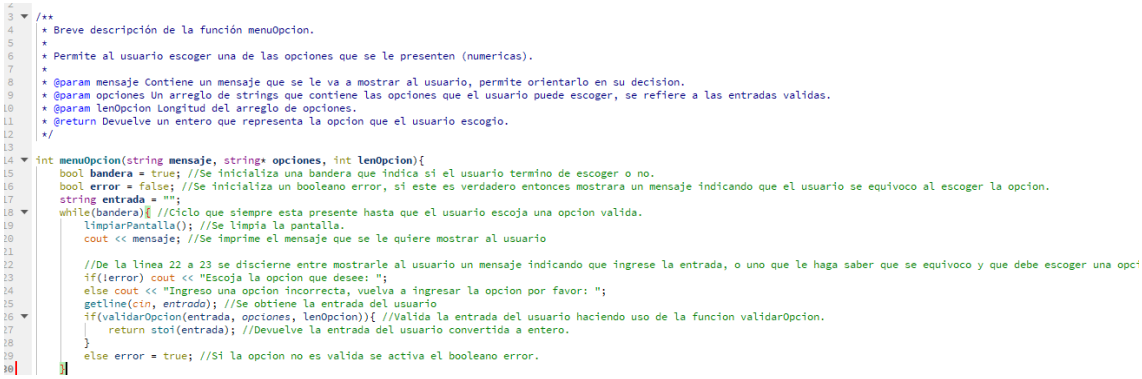
- **Función generarCandado:**

```
int **generarCandado(int *regla, int sizeRegla){
    int intentos = 0; //Se inicializa el contador que cuenta los intentos que se han realizado para obtener una combinación dada.
    int estadoUno = -1; //Define el estado por defecto de la primera matriz a comparar. (se refiere a la primera matriz en cada comparación, no la primera de todas)
    int estadoDos = -1; //Define el estado por defecto de la segunda matriz a comparar. (se refiere a la segunda matriz en cada comparación, no la segunda de todas)
    int **resultado = new int*[sizeRegla - 1]; //Se define la matriz que va a contener los resultados y ordenes de las matrices.
    for(int cont = 0; cont < sizeRegla - 1; cont++) resultado[cont] = new int[2]; //Inicializa cada espacio de la matriz resultado como un arreglo de enteros.
    int ordenMin; //Orden mínimo de cada matriz o estructura
    bool enProceso = true; //Variable que indica si esta en proceso de encontrar una combinacion o no.
    int enteroAux;
    bool boolAux;

    //Lineas de 62 - 70 sirven para determinar el orden mínimo de la matriz segun la regla dada.
    if(regla[0] >= regla[1]){
        ordenMin = regla[0];
    }
    else ordenMin = regla[1];

    if(ordenMin % 2 == 0) ordenMin += 1;
    else ordenMin += 2;

    if(regla[0] == regla[1] && regla[0] == 0) ordenMin = 3;

    int *ordenes = generarOrdenes(ordenMin, sizeRegla-1); //Se genera un arreglo con los ordenes de las matrices en el candado
    int ***estructura = crearEstructura(ordenes, sizeRegla - 1); //Se crea una estructura que puede verse como un arreglo de matrices
    int *reglaFunc = new int[5]; reglaFunc[0] =  regla[0]; reglaFunc[1] = regla[1]; reglaFunc[2] = regla[2]; reglaFunc[3] = 0; reglaFunc[4] = 0; //Es la regla que se necesita para cada combinacion en la funcion encontra
    int *resulAux = new int[2]; //Resultado auxiliar, este arreglo va a ser manejado por la funcion encontrarComb
    int *ordenAux = new int[2]; //Orden del par de matrices que se este comparando
    for(int cont = 0; cont < sizeRegla - 2; cont++){ //Ciclo for que va a coger cada criterio de la regla ingresada y adecuando las matrices para que todas los criterios se cumplan a cabalidad
        //Para cada iteración vuelve el primer elemento del arreglo ordenAux el orden de la primera matriz a comparar, analogamente se deduce el funcionamiento de la linea 80.
        ordenAux[0] = ordenes[cont];
        ordenAux[1] = ordenes[cont+1];
        while(enProceso){ //Ciclo que se ejecuta mientras no se encuentre una combinación valida para dos matrices.
            boolAux = encontrarComb(estructura[cont], estructura[cont+1], reglaFunc, resulAux, ordenAux, estadoUno, estadoDos); //Este booleano indica si se pudo encontrar una combinacion o no.  Use of undeclared …
            if(!boolAux){ //Si no se pudo encontrar una combinación con esas dos matrices se aumentara el orden siguiendo los lineamientos comentados a continuacion:
                if(cont == 0){ //Si nos encontramos en la primera comparacion se genera un numero entre 0 y 1, con el fin de aumentar el orden de cualquiera de las dos matrices a comparar.
                    enteroAux = genRandom(0,1); //Se genera el numero aleatorio
                    //Se editan los ordenes según corresponda.
                    ordenAux[enteroAux] += 2;
                    ordenes[enteroAux] += 2;
                    editarEstructura(enteroAux, ordenAux[enteroAux], estructura); //Finalmente el arreglo de matrices se ve editado, poniendo en la estructura la nueva matriz con el nuevo orden.
                }
                else{ //Si no nos encontramos en la primera comparacion, se sabe que la segunda matriz de la comparación debe ser la que cambie, no la primera, porque si se cambia la primera podriamos alterar la correct
                    else{ //Si no nos encontramos en la primera comparacion, se sabe que la segunda matriz de la comparación debe ser la que cambie, no la primera, porque si se cambia la primera podriamos alterar la correct
                        //Se realiza el mismo proceso que en el if anterior.
                        ordenAux[1] += 2;
                        ordenes[cont + 1] += 2;
                        editarEstructura(cont + 1, ordenAux[1], estructura);
                    }
                }
                else{ //En este bloque de codigo se realizan las ediciones pertinentes a los arreglos que llevan control de los resultados correctos, posteriormente se vuelve falso el booleano enProceso.
                    //Y la primera matriz de la comparación se vuelve estatica.
                    if(cont == 0){
                        if(sizeRegla >= 4){
                            reglaFunc[2] = regla[3]; reglaFunc[3] = 1;
                        }
                        resultado[0][0] = ordenAux[0]; resultado[0][1] = resulAux[0]; resultado[1][0] = ordenAux[1]; resultado[1][1] = resulAux[1];
                    }
                    else resultado[cont + 1][0] = ordenAux[1];  resultado[cont+1][1] = resulAux[1]; reglaFunc[2] = regla[3 + cont];
                    enProceso = false;
                    estadoUno = resulAux[1];
                }
                intentos++; //Se aumenta el contador de intentos
                if(intentos > 50){ //Si hay mas de 50 intentos se edita el valor del apuntador resultado para dejalo como -1, eso nos indicará despues que nuestro algoritmo no pudo encontrar una combinacion.
                    resultado[0][0] = -1;
                    return resultado;
                }
            }
            intentos = 0; //Para cada comparacion se reinicia el contador de intentos y el booleano enProceso se deja verdadero.
            enProceso = true;
        }

        //Finalmente se libera memoria para evitar la fuga de datos.
        delete [] ordenAux;
        delete [] reglaFunc;
        delete [] ordenes;
        delete [] resulAux;
        destruirEstructura(estructura);
        return resultado;
    }
```

La funcion generarCandado es la encargada de formar la configuración de candado que haga cumplir una regla dada, para ello se hace uso de un ciclo for, en la primera iteración se compararan las matrices uno y dos, en este caso se determinó que si no se encontraba una combinación que validara el criterio para estas dos matrices se iba a aumentar el orden de alguna de las dos de manera aleatoria, para las siguientes iteraciones se deja estática la primera matriz a comparar, y la segunda es la que se le modificará el estado o se aumentará su orden, el proceso continua hasta cumplir a cabalidad todas las comparaciones, si no es posible encontrar una combinación para un criterio dado después de 50 intentos se detiene la ejecución de la funcion. Los resultados obtenidos para cada comparación serán almacenados en una matriz, donde las filas corresponderán a cada una de las matrices, tendrá dos columnas, donde la primera corresponde al orden de la matriz obtenida, y la segunda corresponde al estado de aquella matriz. Como se dijo anteriormente si no se encuentra una combinación despues de 50 intentos se detiene la ejecución del programa, pero el resultado que indica que no se encontró tal combinación será un -1 en el espacio del orden de la primera matriz.

Módulo Menús.

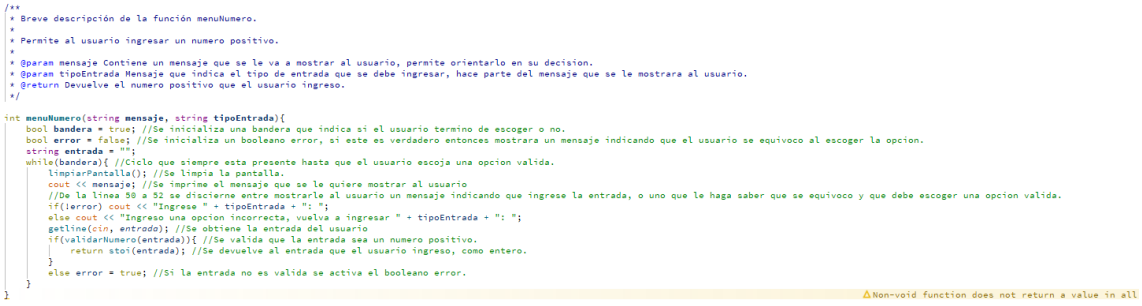
- **Función menuOpcion:**



```
/**
 * Breve descripción de la función menuOpcion.
 *
 * Permite al usuario escoger una de las opciones que se le presenten (numericas).
 *
 * @param mensaje Contiene un mensaje que se le va a mostrar al usuario, permite orientarlo en su decision.
 * @param opciones Un arreglo de strings que contiene las opciones que el usuario puede escoger, se refiere a las entradas validas.
 * @param lenOpcion Longitud del arreglo de opciones.
 * @return Devuelve un entero que representa la opcion que el usuario escogio.
 */

int menuOpcion(string mensaje, string* opciones, int lenOpcion){
    bool bandera = true; //Se inicializa una bandera que indica si el usuario termino de escoger o no.
    bool error = false; //Se inicializa un booleano error, si este es verdadero entonces mostrara un mensaje indicando que el usuario se equivoco al escoger la opcion.
    string entrada = "";
    while(bandera){ //Ciclo que siempre esta presente hasta que el usuario escoja una opcion valida.
        limpiarPantalla(); //Se limpia la pantalla.
        cout << mensaje; //Se imprime el mensaje que se le quiere mostrar al usuario

        //De la linea 22 a 23 se discierne entre mostrarle al usuario un mensaje indicando que ingrese la entrada, o uno que le haga saber que se equivoco y que debe escoger una opci
        if(!error) cout << "Escoja la opcion que desee: ";
        else cout << "Ingreso una opcion incorrecta, vuelva a ingresar la opcion por favor: ";
        getline(cin, entrada); //Se obtiene la entrada del usuario
        if(validarOpcion(entrada, opciones, lenOpcion)){ //Valida la entrada del usuario haciendo uso de la funcion validarOpcion.
            return stoi(entrada); //Devuelve la entrada del usuario convertida a entero.
        }
        else error = true; //Si la opcion no es valida se activa el booleano error.
    }
```

La función menuOpcion recibe lo que digito el usuario en la entrada, se verifica que coincida con las opciones que estableció el programador. Esta también muestra en la consola un mensaje, donde muestra las opciones que se ofrecen. En caso de no cumplir con las condiciones, iterara hasta que el usuario digite la opción correcta.

- **Función menuNumero:**



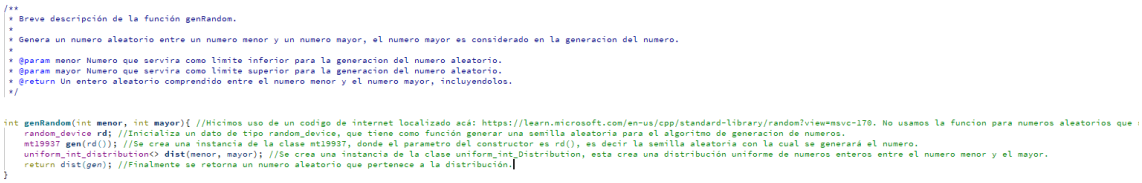
```
/**
 * Breve descripción de la función menuNumero.
 *
 * Permite al usuario ingresar un numero positivo.
 *
 * @param mensaje Contiene un mensaje que se le va a mostrar al usuario, permite orientarlo en su decision.
 * @param tipoEntrada Mensaje que indica el tipo de entrada que se debe ingresar, hace parte del mensaje que se le mostrara al usuario.
 * @return Devuelve el numero positivo que el usuario ingreso.
 */

int menuNumero(string mensaje, string tipoEntrada){
    bool bandera = true; //Se inicializa una bandera que indica si el usuario termino de escoger o no.
    bool error = false; //Se inicializa un booleano error, si este es verdadero entonces mostrara un mensaje indicando que el usuario se equivoco al escoger la opcion.
    string entrada = "";
    while(bandera){ //Ciclo que siempre esta presente hasta que el usuario escoja una opcion valida.
        limpiarPantalla(); //Se limpia la pantalla.
        cout << mensaje; //Se imprime el mensaje que se le quiere mostrar al usuario
        //De la linea 50 a 52 se discierne entre mostrarle al usuario un mensaje indicando que ingrese la entrada, o uno que le haga saber que se equivoco y que debe escoger una opcion valida.
        if(!error) cout << "Ingrese " + tipoEntrada + ": ";
        else cout << "Ingreso una opcion incorrecta, vuelva a ingresar " + tipoEntrada + ": ";
        getline(cin, entrada); //Se obtiene la entrada del usuario
        if(validarNumero(entrada)){ //Se valida que la entrada sea un numero positivo.
            return stoi(entrada); //Se devuelve al entrada que el usuario ingreso, como entero.
        }
        else error = true; //Si la entrada no es valida se activa el booleano error.
    }
}
```

Non-void function does not return a value in all

La función menuNumero cumple la misma funcion que la funcion menuOpcion, pero esta vez le permitirá al usuario ingresar un número, el mensaje que se le va a mostrar al usuario estará dado como parámetro, además de eso también recibe un tipo de entrada, esto con el fin de personalizar el mensaje que se le va a mostrar al usuario. El usuario ingresa una entrada, y si no corresponde con un numero continúa pidiéndoselo hasta que ingrese un numero válido.

Módulo Utilidades.

- **Función genRandom:**



```
/**
 * Breve descripción de la función genRandom.
 *
 * Genera un numero aleatorio entre un numero menor y un numero mayor, el numero mayor es considerado en la generacion del numero.
 *
 * @param menor Numero que servira como limite inferior para la generacion del numero aleatorio.
 * @param mayor Numero que servira como limite superior para la generacion del numero aleatorio.
 * @return Un entero aleatorio comprendido entre el numero menor y el numero mayor, incluyendolos.
 */

int genRandom(int menor, int mayor){ //Hicimos uso de un codigo de internet localizado acá: https://learn.microsoft.com/en-us/cpp/standard-library/random?view=msvc-170. No usamos la funcion para numeros aleatorios que
    random_device rd; //Inicializa un dato de tipo random_device, que tiene como función generar una semilla aleatoria para el algoritmo de generacion de numeros.
    mt19937 gen(rd()); //Se crea una instancia de la clase mt19937, donde el parametro del constructor es rd(), es decir la semilla aleatoria con la cual se generará el numero.
    uniform_int_distribution<> dist(menor, mayor); //Se crea una instancia de la clase uniform_int_Distribution, esta crea una distribución uniforme de numeros enteros entre el numero menor y el mayor.
    return dist(gen); //Finalmente se retorna un numero aleatorio que pertenece a la distribución.
}
```

genRandom permite generar un numero aleatorio entre dos números, uno mayor que otro, la implementación de esta funcion no es propia, el lugar de donde se extrajo está puesto en la descripción de esta, para generar un número aleatorio se crea una semilla aleatoria con la cual el algoritmo mt19937 podrá generar números aleatorios de una distribución de enteros, cabe recalcar que este algoritmo también se trata de generación de números pseudoaleatorios.

- **Función limpiarPantalla:**

```
/**
 * Breve descripción de la función limpiarPantalla.
 *
 * Imprime 50 saltos de linea, simulando que se elimino lo que habia previamente en la pantalla.
 *
 * @return Ninguno, solo son impresiones.
 */

void limpiarPantalla(){
    for(int cont = 0; cont < 50; cont++) cout << "\n";
}
```

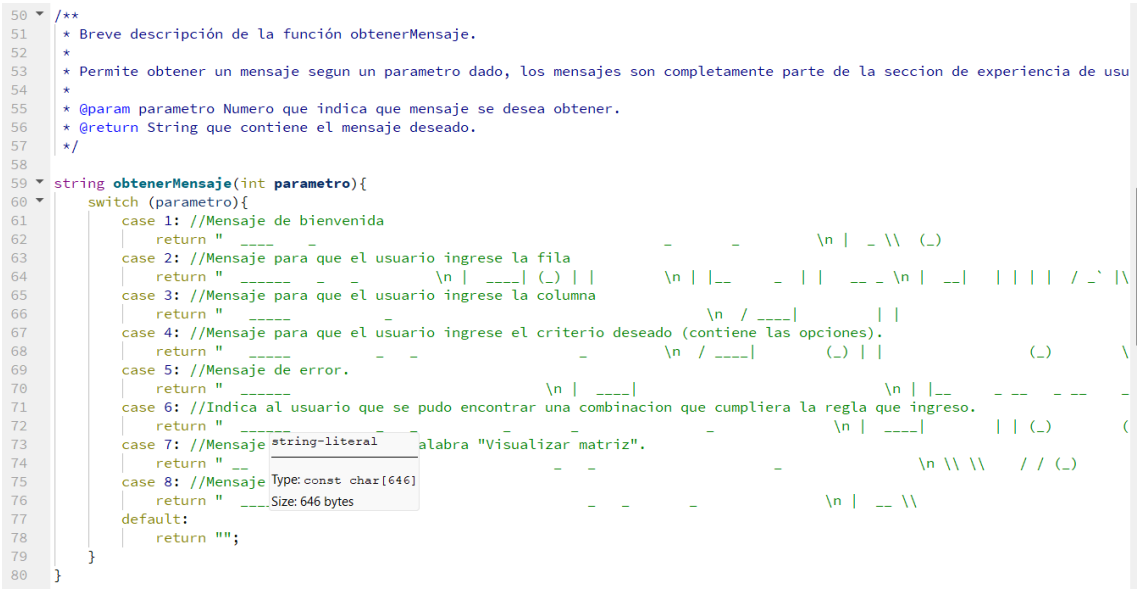
La función limpiarPantalla sirve para que de una apariencia de que la consola este vacía de datos. El truco está en dar 50 saltos de línea.

- **Función stringArray:**

```
/**
 * Breve descripción de la función stringInArray.
 *
 * Verifica si una cadena es un elemento de un arreglo.
 *
 * @param cadena Cadena dada.
 * @param arreglo Arreglo en el cual debe comprobarse si existe una cadena dada o no.
 * @param lenArreglo Tamaño del arreglo dado.
 * @return Booleano que indica si la cadena dada es un elemento del arreglo especificado.
 */

bool stringInArray(string cadena, string* arreglo, int lenArreglo){
    for(int cont = 0; cont < lenArreglo; cont ++) if(cadena == arreglo[cont]) return true;
    return false;
}
```

La función stringArray verifica si una cadena pertenece a un arreglo de strings, esto se hace a partir de un ciclo for que evalúa cada elemento del arreglo comparándolo con la cadena, si encuentra ocurrencias retorna verdadero, sino retorna falso.

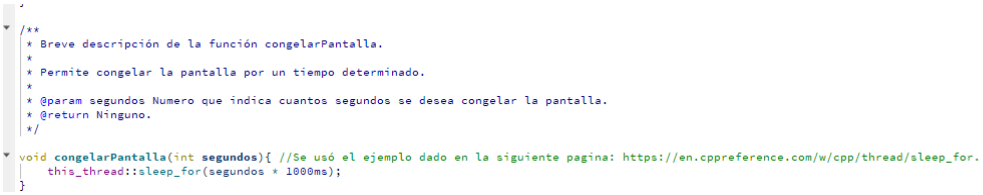
- **Función obtenerMensajes:**



```
/**
 * Breve descripción de la función obtenerMensaje.
 *
 * Permite obtener un mensaje segun un parametro dado, los mensajes son completamente parte de la seccion de experiencia de usu
 *
 * @param parametro Numero que indica que mensaje se desea obtener.
 * @return String que contiene el mensaje deseado.
 */

string obtenerMensaje(int parametro){
    switch (parametro){
        case 1: //Mensaje de bienvenida
            return "  ____    _                                          _       _         \n |  _ \\  (_)
        case 2: //Mensaje para que el usuario ingrese la fila
            return "  ______   _   _          \n |  ____| (_) | |        \n | |__     _  | |   __ _ \n |  __|   | | | |  / _` |\
        case 3: //Mensaje para que el usuario ingrese la columna
            return "   _____           _                                \n  / ____|         | |
        case 4: //Mensaje para que el usuario ingrese el criterio deseado (contiene las opciones).
            return "   _____       _   _                    _         \n  / ____|        (_) | |                 (_)        \
        case 5: //Mensaje de error.
            return "  ______                                  \n |  ____|                              \n |__     _ __    _ __    _
        case 6: //Indica al usuario que se pudo encontrar una combinacion que cumpliera la regla que ingreso.
            return "  ______   _   _          _       _              _              \n |  ____|        | | (_)        (
        case 7: //Mensaje
            return "  __                                    _   _                    _                \n \\ \\  \\    / / (_)
        case 8: //Mensaje
            return "  ____                                      _   _       _              \n |  __ \\
        default:
            return "";
    }
}
```


La función obtenerMensajes retorna un mensaje que despues se va a mostrar en consola. Se establece un parámetro, y dependiendo del este se devolverá un mensaje u otro.

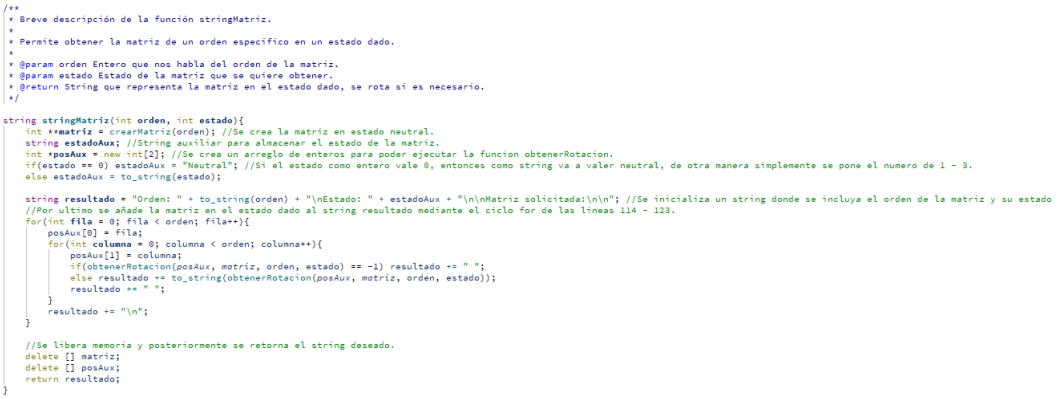
- **Función congelarPantalla:**



```
/**
 * Breve descripción de la función congelarPantalla.
 *
 * Permite congelar la pantalla por un tiempo determinado.
 *
 * @param segundos Numero que indica cuantos segundos se desea congelar la pantalla.
 * @return Ninguno.
 */

void congelarPantalla(int segundos){ //Se usó el ejemplo dado en la siguiente pagina: https://en.cppreference.com/w/cpp/thread/sleep_for.
    this_thread::sleep_for(segundos * 1000ms);
}
```

La función congelarPantalla hace uso de la biblioteca thread para congelar la pantalla unos segundos dados, esta funcion fue extraída de la página que se cita en la descripción de la función.

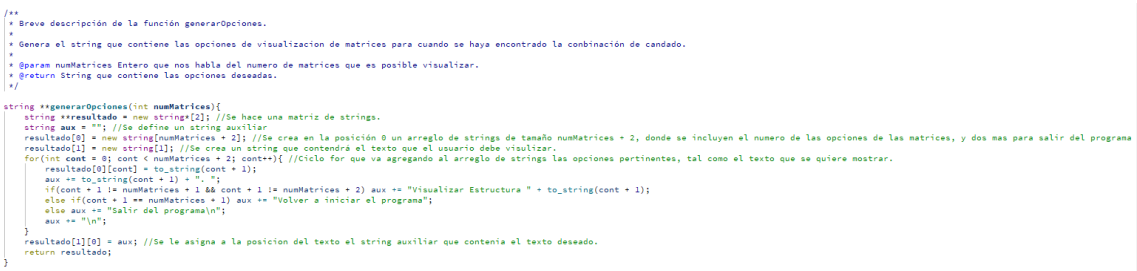
- **Función stringMatriz:**



```
/**
 * Breve descripción de la función stringMatriz.
 *
 * Permite obtener la matriz de un orden especifico en un estado dado.
 *
 * @param orden Entero que nos habla del orden de la matriz.
 * @param estado Estado de la matriz que se quiere obtener.
 * @return String que representa la matriz en el estado dado, se rota si es necesario.
 */

string stringMatriz(int orden, int estado){
    int **matriz = crearMatriz(orden); //Se crea la matriz en estado neutral.
    string estadoAux; //String auxiliar para almacenar el estado de la matriz.
    int *posAux = new int[2]; //Se crea un arreglo de enteros para poder ejecutar la funcion obtenerRotacion.
    if(estado == 0) estadoAux = "Neutral"; //Si el estado como entero vale 0, entonces como string va a valer neutral, de otra manera simplemente se pone el numero de 1 - 3.
    else estadoAux = to_string(estado);

    string resultado = "Orden: " + to_string(orden) + "\nEstado: " + estadoAux + "\n\nMatriz solicitada:\n\n"; //Se inicializa un string donde se incluya el orden de la matriz y su estado
    //Por ultimo se añade la matriz en el estado dado al string resultado mediante el ciclo for de las lineas 114 - 123.
    for(int fila = 0; fila < orden; fila++){
        posAux[0] = fila;
        for(int columna = 0; columna < orden; columna++){
            posAux[1] = columna;
            if(obtenerRotacion(posAux, matriz, orden, estado) == -1) resultado += " ";
            else resultado += to_string(obtenerRotacion(posAux, matriz, orden, estado));
            resultado += " ";
        }
        resultado += "\n";
    }

    //Se libera memoria y posteriormente se retorna el string deseado.
    delete [] matriz;
    delete [] posAux;
    return resultado;
}
```

La función stringMatriz permite obtener un string que contiene a una matriz de orden dado, en un estado dato, sirve para mostrarle al usuario una matriz en un estado dado, es utilizada cuando se genera la configuración de candado y el usuario debe escoger cual matriz visualizar.

- **Función generarOpciones:**

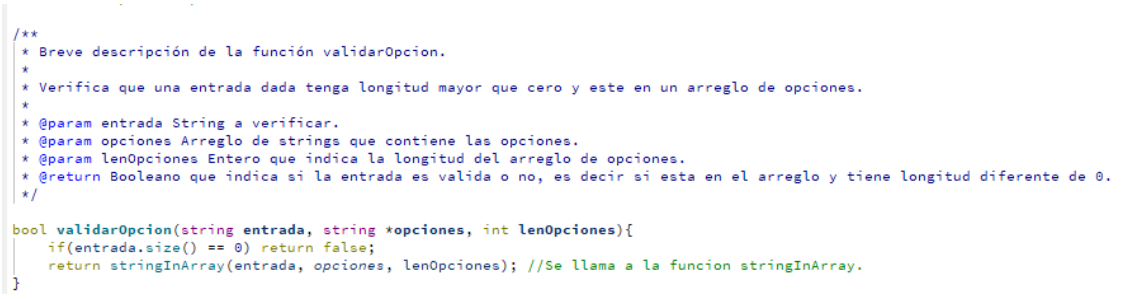


```
/**
* Breve descripción de la función generarOpciones.
*
* Genera el string que contiene las opciones de visualizacion de matrices para cuando se haya encontrado la conbinación de candado.
*
* @param numMatrices Entero que nos habla del numero de matrices que es posible visualizar.
* @return String que contiene las opciones deseadas.
*/

string **generarOpciones(int numMatrices){
    string **resultado = new string*[2]; //Se hace una matriz de strings.
    string aux = ""; //Se define un string auxiliar
    resultado[0] = new string[numMatrices + 2]; //Se crea en la posición 0 un arreglo de strings de tamaño numMatrices + 2, donde se incluyen el numero de las opciones de las matrices, y dos mas para salir del programa
    resultado[1] = new string[1]; //Se crea un string que contendrá el texto que el usuario debe visulizar.
    for(int cont = 0; cont < numMatrices + 2; cont++){ //Ciclo for que va agregando al arreglo de strings las opciones pertinentes, tal como el texto que se quiere mostrar.
        resultado[0][cont] = to_string(cont + 1);
        aux += to_string(cont + 1) + ". ";
        if(cont + 1 != numMatrices + 1 && cont + 1 != numMatrices + 2) aux += "Visualizar Estructura " + to_string(cont + 1);
        else if(cont + 1 == numMatrices + 1) aux += "Volver a iniciar el programa";
        else aux += "Salir del programa\n";
        aux += "\n";
    }
    resultado[1][0] = aux; //Se le asigna a la posicion del texto el string auxiliar que contenia el texto deseado.
    return resultado;
}
```

La función generarOpciones es útil para crear un mensaje que contiene las matrices correspondientes a una configuración de candados, este mensaje va a contener las opciones disponibles que tiene el usuario, entre ellas estan todas las matrices que surgieron de la creación del candado y por último una opción para volver a iniciar el programa o salir del mismo, esta función a su vez crea un arreglo con las opciones disponibles, y sirve para ingresar estas opciones a la funcion menuOpcion.

Módulo Validacion.

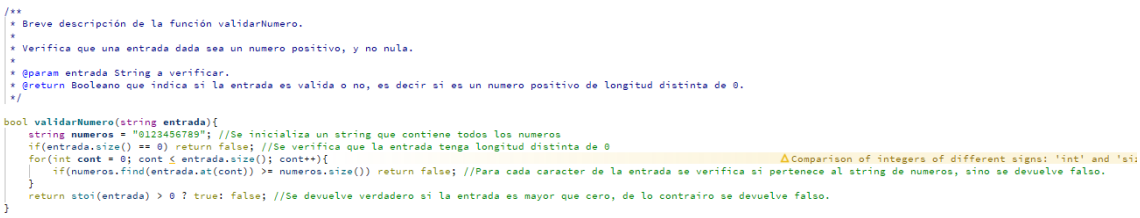
- **Función validarOpcion:**



```
/**
 * Breve descripción de la función validarOpcion.
 *
 * Verifica que una entrada dada tenga longitud mayor que cero y este en un arreglo de opciones.
 *
 * @param entrada String a verificar.
 * @param opciones Arreglo de strings que contiene las opciones.
 * @param lenOpciones Entero que indica la longitud del arreglo de opciones.
 * @return Booleano que indica si la entrada es valida o no, es decir si esta en el arreglo y tiene longitud diferente de 0.
 */

bool validarOpcion(string entrada, string *opciones, int lenOpciones){
    if(entrada.size() == 0) return false;
    return stringInArray(entrada, opciones, lenOpciones); //Se llama a la funcion stringInArray.
}
```

Esta funcion valida si una entrada tiene longitud mayor que cero, y además de eso verifica si la entrada está en un arreglo de opciones válidas, si lo está devuelve true, sino devuelve false.

- **Función validarNumero:**



```
/**
 * Breve descripción de la función validarNumero.
 *
 * Verifica que una entrada dada sea un numero positivo, y no nula.
 *
 * @param entrada String a verificar.
 * @return Booleano que indica si la entrada es valida o no, es decir si es un numero positivo de longitud distinta de 0.
 */

bool validarNumero(string entrada){
    string numeros = "0123456789"; //Se inicializa un string que contiene todos los numeros
    if(entrada.size() == 0) return false; //Se verifica que la entrada tenga longitud distinta de 0
    for(int cont = 0; cont < entrada.size(); cont++){                    ⚠ Comparison of integers of different signs: 'int' and 'si
        if(numeros.find(entrada.at(cont)) >= numeros.size()) return false; //Para cada caracter de la entrada se verifica si pertenece al string de numeros, sino se devuelve falso.
    }
    return stoi(entrada) > 0 ? true: false; //Se devuelve verdadero si la entrada es mayor que cero, de lo contrairo se devuelve falso.
}
```

Valida si cada carácter de una entrada pertenece a un string de números, y además de eso si la longitud de la entrada es mayor que cero, si lo dicho anteriormente se cumple se devuelve true, en caso contrario se devuelve false.

Problemas de desarrollo afrontados.

Durante el desarrollo de la solución afrontamos dos problemas que pudimos resolver con solvencia, el principal problema que afrontamos fue deducir cómo obtener el valor de una celda después de una rotación, lo que sucede es que nosotros no vimos factible rotar la matriz para cada comparación que íbamos a hacer, por lo tanto, mediante tanteo dedujimos una relacion entre el centro de la matriz y el valor de la celda dada después de efectuar cualquier rotación, sea en el estado que sea, esta relacion no fue descubierta sino despues de un par de horas de comenzar a indagar acerca de la misma, después de encontrar la relación la implementación fue relativamente facil.

El segundo problema estuvo relacionado con la experiencia de usuario, para nosotros fue complicado realizar un menú que le permitiera al usuario escoger una opción y a su vez simular que el usuario iba navegando entre menús, despues de varios días intentando crear las funciones se pudo resolver mediante dos funciones que iban a administrar cualquier menú que necesitemos, una de ellas fue la función menuOpcion que funcionaba solo para recibir opciones, y otra menuNumero, que recibía solo números, a su vez creamos distintas funciones que nos permitieron darle estética al codigo, tales como congelarPantalla() o limpiarPantalla(), haciendo uso de una página web generamos ASCII Art para cada menú que íbamos a mostrar. Inclusive pudimos hacer que el codigo pudiera iniciar de nuevo si el usuario lo desea, en nuestra opinión resolvimos de buena manera el problema.

Los problemas que afrontamos fueron realmente pocos, esto debido a que teníamos claros los conceptos que íbamos a utilizar en cada parte del codigo, entre ellos estuvieron arreglos, punteros, memoria dinámica, entre otros.

Evolución de la solución.

- **26 de marzo:**
    - Se inicio la creación del programa. En dicho día, se crearon el módulo estructura, donde están las funciones básicas como: crearMatriz y crearEstructura. Que serían implementados en un main, para ser probadas.
    - Se añaden dos funciones más en el módulo Estructura: obtenerRotacion y destruirEstructura. Además, ya se crean las primeras funciones para el módulo candado, como son: validarRegla, generarCandado, primeraComb (Lo que sería más adelante encontrarComb). En el main, se hacen modificaciones para probar obtenerRotacion.
    - Se crea la función encontrarComb, implementándose en el main para encontrar las posibles combinaciones que existen entre dos matrices. Dicha función está en el módulo candado. Lo que hace es verificar si se cumple o no una regla en una celda, además de si la matriz va a rotar o no.
    - Se elimina la función de primeraComb. Además de que la función generarCandado se completó, y esta implementa las funciones conocidas, para que el programa funcione a partir de una regla dada.
- **27 de marzo:**
    - Se modifico en la función validarRegla una línea de código, que comprueba si el valor de la fila y la columna es mayor a cero, dicha modificación fue cambiar un and y se puso un or. Y por último se modificó en la función encontrarComb: se hizo que en vez de asignar continuamente valores a celdauno y celdados, solo se le asignara una vez.
    - Se empezaron a crear los módulos que tienen que ver con la experiencia de usuario, es decir: Los módulos menús, utilidades y validación.
- **28 de marzo:**
    - Se empezó subió el avance del informe, y se organizó el código por carpetas.
- **30 de marzo:**
    - Implementado varios módulos, entre esos menus, el usuario ya puede ingresar los datos de la regla. Además, se corrigió un error en la función encontrarComb: El error consiste en que, si no hallaba una combinación, iteraba infinitamente.
    - Se escribió a modo de ayuda, para orientarnos en el código docstrings, en las líneas pertinentes de cada módulo.
- **1° de abril:**
    - Se continúa escribiendo dosctrings en las líneas de código pertinentes que faltan. Además de que se escribe en el cuerpo de cada función, para que sirve la misma.
- **7° de abril:**
    - Se agregan unos pocos detalles al codigo, faltando solamente realizar el video e insertar el informe en formato PDF.
- **8° de abril:**
    - Se dio por finalizado el codigo, insertando así el informe y el video requerido.