**Fundamentos de Programación (9 de septiembre de 2016)**

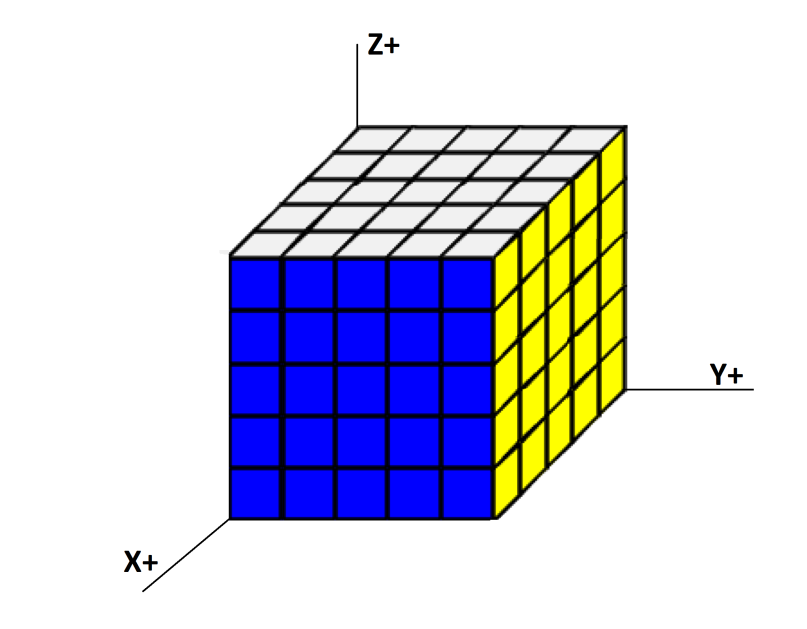
**(Grados en Ingeniería Mecánica, Eléctrica, Electrónica Industrial y Química Industrial)**

***Ejercicio:*** construir un programa en ***C*** lo más modular posible (atendiendo a los criterios de modularidad) que se ajuste a la especificación dada y documentar el diseño preliminar con la definición de las nuevas tipologías de datos, el diagrama de módulos (estructura del programa) y las interfaces de los módulos, y el diseño detallado con las definiciones de los respectivos sub-programas.

**Ejercicio:** Construir un programa para simular los movimientos de giro en la resolución de un cubo o puzzle de ***Rubik*** de longitud de lado ***n*** unidades. Un cubo de ***Rubik*** podemos considerarlo como un cubo grande compuesto en principio por ***n3*** cubos unitarios ó piezas, que en el estado ordenado cada cara del cubo es de un color. Existen 4 tipos de piezas según su posición relativa en el cubo: vértices (***8***), aristas (***12\*(n-2)***), centros (***6\*(n-2)2***) e interiores (***(n-2)3***). Los vértices y aristas si tienen realmente forma de cubo, con una particular pestaña de sujeción. Los centros son en realidad medios cubos y están sujetos mediante un eje a un sistema de crucetas centrales (mecanismo interno que permite las posibilidades de giro de las caras). Los interiores son los que conforman dicho mecanismo interno y no son visibles.

En el ejemplo se muestra un cubo de ***Rubik*** de lado ***5***, con un sistema de referencia inicial para identificar sus caras, junto con la nomenclatura de las mismas y su configuración de colores en el estado ordenado inicial:

|  |  |  |  |
| --- | --- | --- | --- |
| **Cara** | **Vector normal** | **Nomenclatura** | **Color inicial** |
| 0 | X+ | Frontal | A: azul |
| 1 | Y+ | Derecha | M: amarillo |
| 2 | Z+ | Arriba | B: blanco |
| 3 | X- | Trasera | N: negro |
| 4 | Y- | Izquierda | R: rojo |
| 5 | Z- | Abajo | V: verde |



Capa n-1

...

Eje Z+ ...

...

Capa 1

Capa 0

Capa 0

Capa 1

...

...

...

Capa n-1

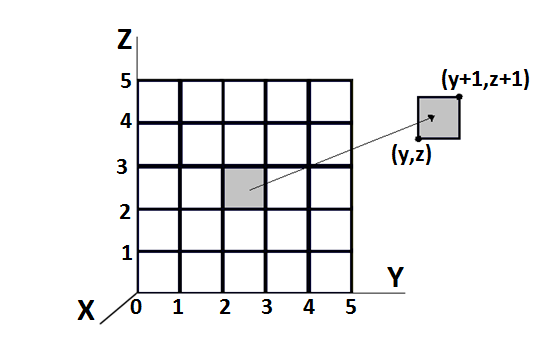
Eje X+ Capa 0

Capa1

...

Capa n-1

Eje Y+



Un detalle importante es que las piezas unitarias sólo tienen pegatinas de colores en las caras visibles inicialmente lo que permite simplificar la información a representar. Para identificar la posición de una cara visible de una pieza dentro del plano a la que pertenece, se utilizarán las coordenadas de intersección de línea de cuadrícula de una de sus esquinas. Ejemplo: ***n=5***, Plano ***X+*** (ó ***X-***):

Nota: para los otros planos, aplicar simetría mediante el intercambio de los ejes en orden: X🡪Y🡪Z🡪X (0🡪1🡪2🡪0)

El movimiento básico del cubo de ***Rubik*** es el giro de ***90º*** alrededor de un eje paralelo a uno de los ejes cartesianos y que pasa por el centro del cubo. Con respecto a dicho eje de rotación, podemos considerar el cubo dividido en ***n*** niveles ó capas perpendiculares al eje de rotación. Un movimiento del cubo vendría por tanto descrito por los siguientes parámetros:

* Eje de rotación: éste atraviesa el centro del cubo y solo se admiten tres posibilidades: 0: paralelo a X+, 1: paralelo a Y+, 2: paralelo a Z+.
* Capa o nivel: nº entero en el intervalo ***[0,n-1]*** que representa al plano de cubos unitarios perpendicular al eje de rotación y numerado según el sistema de referencia inicial.
* Ángulo de giro: debe de ser un múltiplo entero de ***90º*** (positivo en sentido anti-horario con respecto al eje de rotación). Nótese que un giro de ***180º*** equivale a dos giros consecutivos de ***90º***, y que un giro de ***270º*** (ó ***-90º***) es igual a tres giros consecutivos de ***90º***.

Ejemplo: ***n=5***, **giro 90º, capa 1, eje X+**

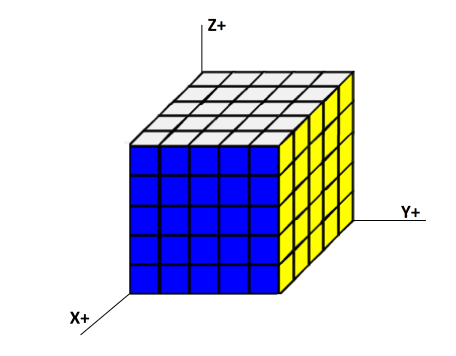
|  |  |  |  |
| --- | --- | --- | --- |
| **Estado inicial** | **Capa 1 eje X+** | **Giro 90º capa 1** | **Estado final** |
|  |  |  |  |

Nota: para giros alrededor de otro eje, aplicar simetría mediante el intercambio de los ejes en orden: X🡪Y🡪Z🡪X (0🡪1🡪2🡪0)

El programa solicitará inicialmente por teclado la longitud ***n*** del lado del cubo de ***Rubik*** en piezas (mínimo ***1*** y máximo ***10***). Seguidamente inicializará los colores de las caras visibles de las piezas con sus correspondientes valores en su estado ordenado inicial, e imprimirá dicho estado en pantalla simulando una proyección paralela axonométrica de las caras ocultas y visibles del cubo, según el formato indicado en el siguiente ejemplo:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Caras ocultas** | |  | **Caras visibles** | |
| **Cara**  **Y-**    **X+** | **Z+**  **Cara X-**  **Y+** | **Cara**  **Y- Z+** | **X-**  **Cara**  **Y+**    **Z-** |
| **Cara**  **Z-** | **Cara X+** |

|  |
| --- |
| Caras ocultas: Caras visibles:    /| N N N N N | / B B B B B/M|  /R| N N N N N | / B B B B B/MM|  /RR| N N N N N | / B B B B B/MMM|  /RRR| N N N N N | / B B B B B/MMMM|  /RRRR| N N N N N | / B B B B B/MMMMM|  |RRRRR/ V V V V V/ |A A A A A |MMMM/  |RRRR/ V V V V V/ |A A A A A |MMM/  |RRR/ V V V V V/ |A A A A A |MM/  |RR/ V V V V V/ |A A A A A |M/  |R/ V V V V V/ |A A A A A |/ |
| 01234567890123456789012345678901234567890123456789 |



A continuación, se realizarán ***20*** giros aleatorios para configurar el cubo en un estado desordenado preparado para su resolución, presentándose el mismo en pantalla. Finalmente, para resolver el cubo se repetirán de forma indefinida los dos siguientes pasos hasta que el cubo se encuentre en un estado ordenado:

* Seleccionar un movimiento de giro, mediante la introducción por teclado del eje de giro (entero en intervalo ***[0,2]***), la capa ó nivel a girar (entero en intervalo ***[0,n-1]***) y el ángulo de giro (entero en intervalo ***[1,3]*** que representa el nº de giros sucesivos de ***90º*** a realizar).
* Imprimir en pantalla el estado resultante del movimiento seleccionado.

|  |  |
| --- | --- |
| Funciones para generar números aleatorios en C | |
| #include <stdlib.h>  #include <time.h> |  |
| time\_t t;  srand((unsigned) time(&t)); | Inicializa el generador de números aleatorios con un valor aleatorio obtenido del reloj del sistema |
| rand()%(b-a+1)+a | Genera un número aleatorio entero en intervalo [a,b]=a,a+1,a+2,...,b-1,b |
| a+(b-a)\*1.0\*rand()/RAND\_MAX | Genera un número aleatorio real en intervalo [a,b] |