**- Array o Arreglo -**

     Un **array** es un identificador que referencia un conjunto de datos del mismo tipo. Imagina un tipo de dato **int**; podremos crear un conjunto de datos de ese tipo y utilizar uno u otro con solo cambiar el índice que lo referencia. El índice será un valor entero y positivo. En C++ los *arrays*comienzan por la posición **0**.

**Vectores:**

     Un **vector** es un *array* *unidimensional*, es decir, solo usa un índice para referenciar a cada uno de los elementos.

     Su declaración será:         **tipo nombre [tamaño];**

     El tipo puede ser cualquiera de los ya conocidos y el tamaño indica el número de elementos del vector (se debe indicar entre corchetes **[ ]**). En el ejemplo puedes observar que la variable **i** es utilizada como índice, el primer **for** sirve para rellenar el vector y el segundo para visualizarlo. Como ves, las posiciones van de **0** a **9** (total 10 elementos).

     (Tomado de "Introducción al [lenguaje de programación](http://www.monografias.com/Computacion/Programacion/) de C/C++". Sergio Pacho)

     Ejemplo:

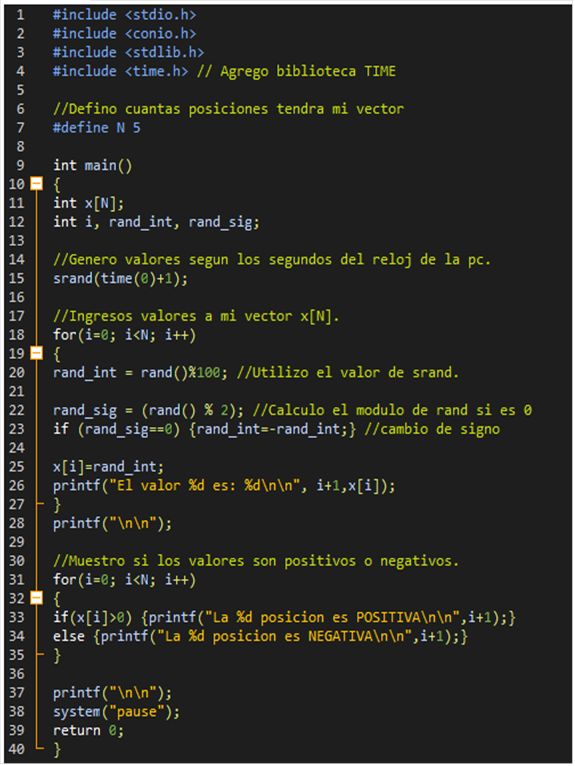
**int** **num** [**100**]; /\*Arreglo de tipo entero compuesto de 100 posiciones\*/

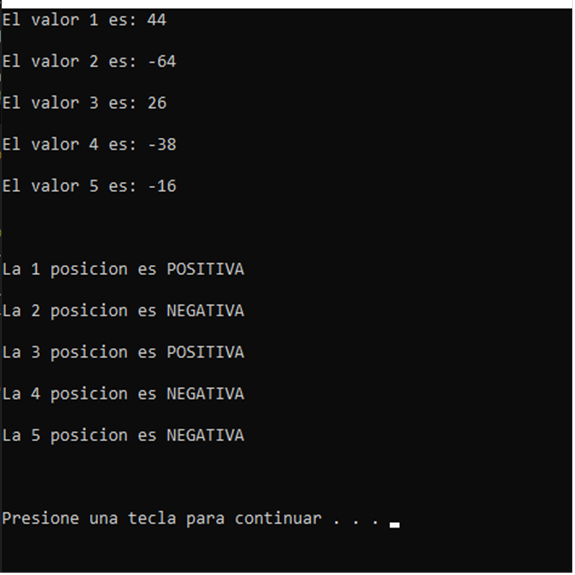
**char** **nom** [**80**]; /\*[Texto](http://www.monografias.com/trabajos13/libapren/libapren.shtml) de 80 caracteres\*/

**float** **x**[**12**]; /\*arreglo de 12 elementos punto flotantes \*/

**Ejemplo Vector 1:**

     A continuación, se muestra un programa que lee un vector de 5 posiciones, y luego determina si la posición es positiva o negativa. También utilizamos un #define denominado “Constante Simbólica” hace más sencillo o más fácil modificar un programa que utiliza arreglos. Ya que todas las referencias al tamaño del arreglo pueden ser alteradas, cambiando el valor de la constante simbólica en nuestro ejemplo de N=”5”.



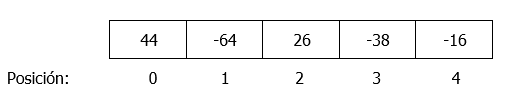


     En el ejemplo estamos haciendo uso de la constante simbólica, de cuyos beneficios ya se habló. La definimos con 5 posiciones, recordando que C++, empieza a contar desde cero. Luego definimos el vector llamado x, como entero, y ojo, que éste va acompañado de su dimensión [N]. Haciendo uso de un for, empezamos a llenar el vector, aquí utilizaremos una función muy interesante que permite generar de números de forma aleatoria es decir randomizar los ingresos, la misma se llama “rand();” para que el ingreso sea totalmente arbitrario voy a asignarle un inicio a rand y para ello voy a utilizar “srand” que captura el tiempo en fecha/hora, este valor cambia si ejecutamos el programa en distinto instante de tiempo, para que sea igual tendríamos que arrancar el programa dos veces en el mismo segundo para obtener la misma secuencia de números aleatorios. La línea de código es: rand\_int = rand()%100;    en este caso le asigno el numero aleatorio a rand\_int y voy a limitar las posibilidades de números del 0 a 99 como se ve en la parte subrayada.

     Pues bien por ultimo voy a utilizar otro rand para obtener símbolos + y -, utilizo para ello el operador “%2” modulo que me calcula si el resto de una división es 1 o 0, la línea de código es: rand\_sig = (rand() % 2);  si es par = 0 entonces con un if le voy a asignar un signo negativo a mi ecuación if (rand\_sig==0) {rand\_int=-rand\_int;}.

     Por ultimo muestro todos mis resultados por pantalla, utilizando un for para recorrer el vector x[N] y un if para mostrar si el resultado es positivo o negativo. Tengan en consideración que para escribir la posición utilice el código:  printf("La %d posicion es …..nn",i+1) tuve que sumarle 1 a i para que por pantalla me muestre que “La posición 1 es…” de otra forma la primer posición hubiese sido 0. Esto ya lo habíamos comentado en clases pasadas pero vale la penar volver a mencionarlo, estos detalles son los que separan buenas prácticas de programación de las malas prácticas que suelen tener aquellos que no se profesionalizan en el tema.

     El siguiente grafico demuestra que sucedio al llenar nuestro vector con los valores indicados. Como podemos ver, C empieza a enumerar las casillas desde el cero, hasta el 4 totalizando de esa manera 5 posiciones. Es por ello, que el for, lo inicializamos con cero, hasta un valor menor que el de la constante del vecto, ya que de lo contrario nos daría un error.



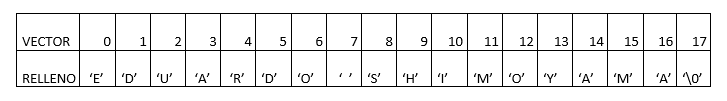
**Vector con caracteres.**

     Una particularidad con los [vectores](http://www.monografias.com/trabajos35/vectores/vectores.shtml) de **tipo char** (cadena de caracteres), es que el elemento que se encuentra el fin de la cadena mediante es un [carácter](http://www.monografias.com/trabajos34/el-caracter/el-caracter.shtml) nulo ().

     Debemos definir una variable de tipo vector con elementos de tipo char:

char programador[18]="EDUARDO SHIMOYAMA";

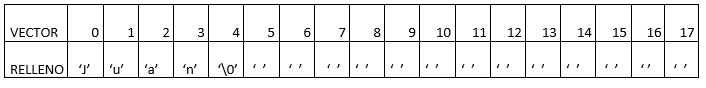
     El tamaño 18 no es un capricho, si contamos la cantidad de caracteres que hay entre las comillas incluyendo el espacio en blanco veremos que es 17. Luego recordar que debe haber un caracter extra en el vector donde se indica que ahí finalizan los datos de la cadena. No hay problema si el vector reserva más de 16 caracteres, en cambio se generarán errores inesperados si el vector tiene menos de 17 caracteres.



     En memoria se almacenan en cada posición del vector el caracter respectivo y cuando se ejecuta el programa y se muestran los datos mediante la función printf se muestran todos hasta que encuentra el caracter NULL que coincide con el caracter ASCII 0.

     Si modificamos el programa y solo guardamos "Juan:" sin modificar el tamaño del vector luego en memoria tenemos:

char programador[18]="Juan";



     Lo que se almacena luego del caracter  se lo considera basura de memoria (puede haber cualquier caracter ASCII).

     Para imprimir un vector de caracteres con la función printf debemos indicar en la cadena de formato el caracter s (string):

printf("%s",programador);

* **Uso de Vectores dentro de las Funciones**

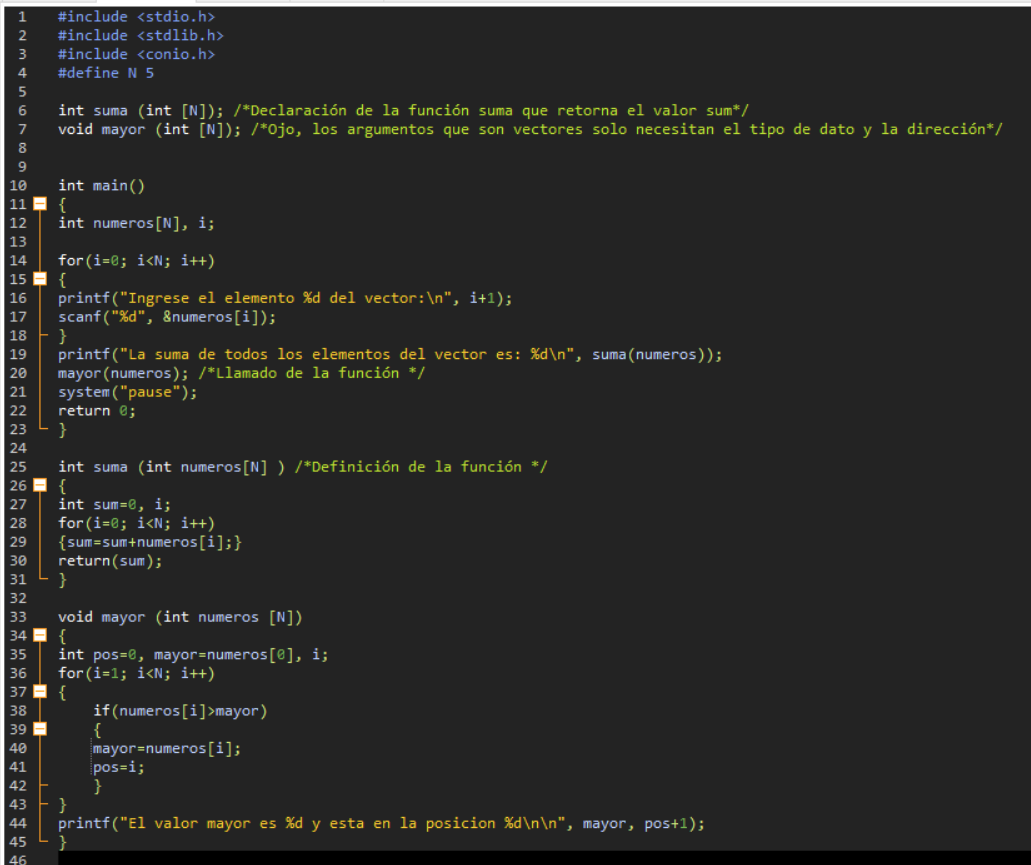
         Un vector, solo puede ser argumento formal, es decir; por el momento, no podemos enviarlo como valor de retorno, se dice por el momento por que cuando hablemos de punteros, veremos que si se pueden enviar.

**Ejemplo Vector 2:**

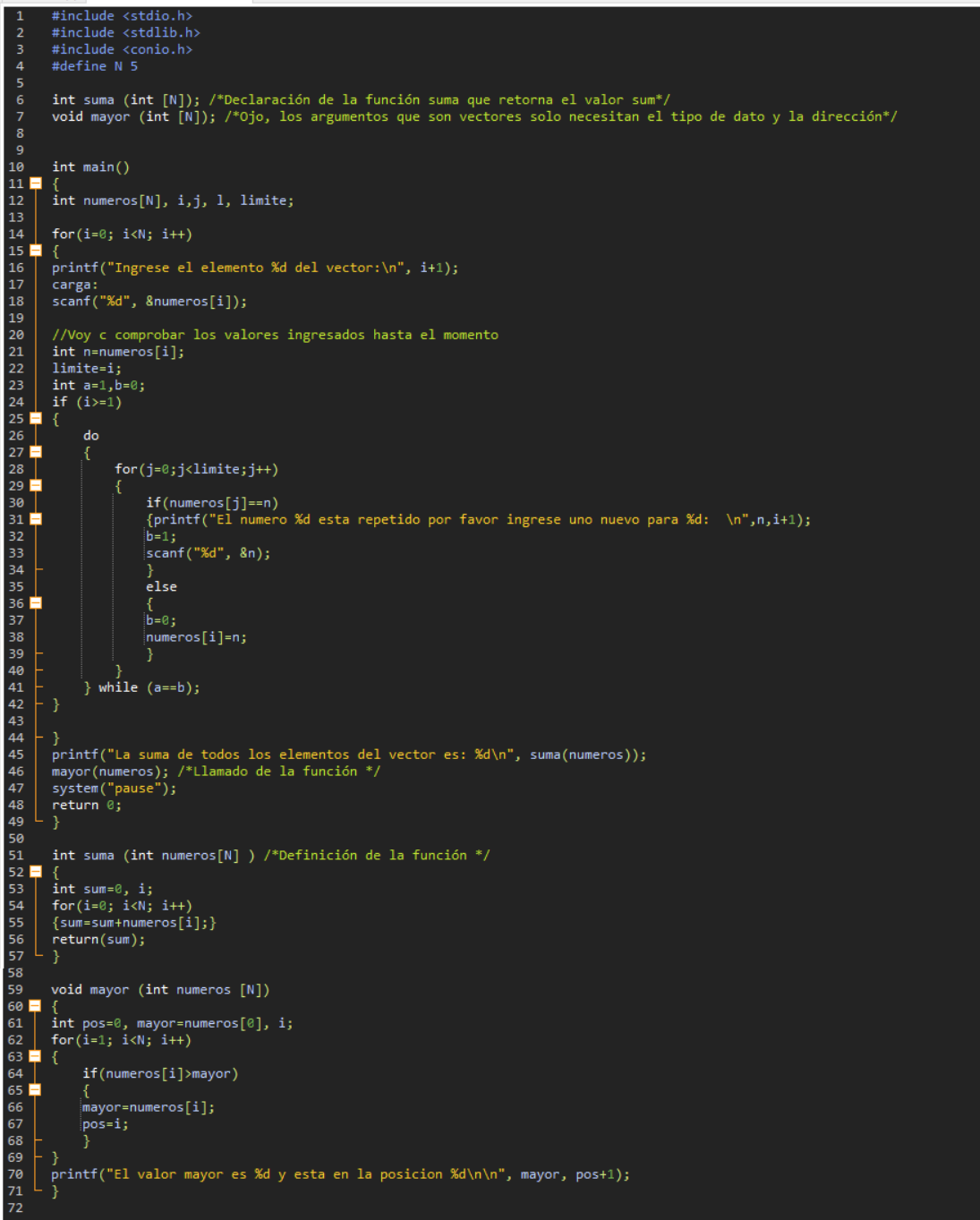
      Diseñe un programa en C, que lea un vector de un máximo de 20 posiciones, y luego determine:

     -La suma de todos los valores.

     -El mayor de los valores, así como la posición del mismo se sabe que dichos datos son de tipo entero.



     El programador, debe preguntarse, ¿qué pasaría si existen dos valores exactamente iguales y que estén en diferentes posiciones, que solución le darías como programador?... este tipo de preguntas deben realizarse siempre que ha finalizado correctamente un programa, y nunca dejar nada sin resolver, porque recordemos que los programas que diseñamos son para que otras personas los utilicen, pueden resolver el problema ustedes?.



         Para solucionar el problema planteado anteriormente voy a comprobar los valores ingresados mediante un for que estará inmerso en un ciclo do while, para validar la salida, es decir siempre que se ingrese un número que ya existe entrara en el ciclo evitando que se repita hasta que este sea diferente a cualquier a de los números ingresados.

**- MATRICES -**

     Las ***matrices***se declaran de forma análoga, con corchetes independientes para cada subíndice. Piensen en la matriz como una hoja de cálculo donde cada número funciona como una dirección de FILA y COLUMNA. La forma general de la declaración es:

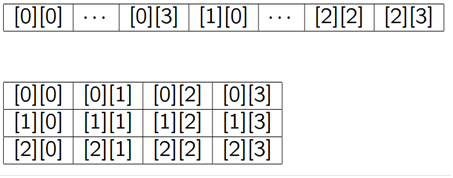
**tipo** **nombre**[**numero\_filas**][**numero\_columnas**];

     Tanto las *filas*como las *columnas*se numeran también a partir de 0. La forma de acceder a los elementos de la [**matriz**](http://www.monografias.com/trabajos10/macroecon/macroecon.shtml) es utilizando su nombre, seguido de las expresiones enteras correspondientes a los dos subíndices, entre corchetes.

     En C++ tanto los vectores como las [**matrices**](http://www.monografias.com/trabajos14/matriz-control/matriz-control.shtml) admiten los *tipos*de las variables escalares (***char***, ***int***, ***long***, ***float***, ***double***, etc.).

     Las matrices en C *se almacenan por filas*, en posiciones consecutivas de [**memoria**](http://www.monografias.com/trabajos13/memor/memor.shtml). En cierta forma, una matriz se puede ver como un *vector de vectores-fila*. Si una matriz tiene N filas (numeradas de 0 a N-1) y M columnas (numeradas de 0 a la M-1).

     Disposicion de una matriz en memoria: int dos\_dim[3][4];



     Como se observa en la primera imagen la estructura de la matriz es lineal, es decir los datos se cargan en forma consecutiva, pero nosotros podemos visualizarla como en la imagen 2, donde el primer dato serán la cantidad de filas y segundo dato la cantidad de columnas.

**Ejemplo Matriz 1:**

         Supongamos que queremos diagramar las tablas de multiplicar del 1 al 9 (¡CUIDADO! Se ha omitido la primera línea de 0 por eso el for inicia con i=1, j =1), para la siguiente demostración una matriz es perfecta, generamos dos líneas del 1 al 9 y hacemos que sus valores se multipliquen entre si, de esta forma en la intersección de coordenadas tendremos el valor de multiplicar el número de fila por el número de columna, asi por ejemplo en la coordenada 2,2 tendremos el valor = 4.

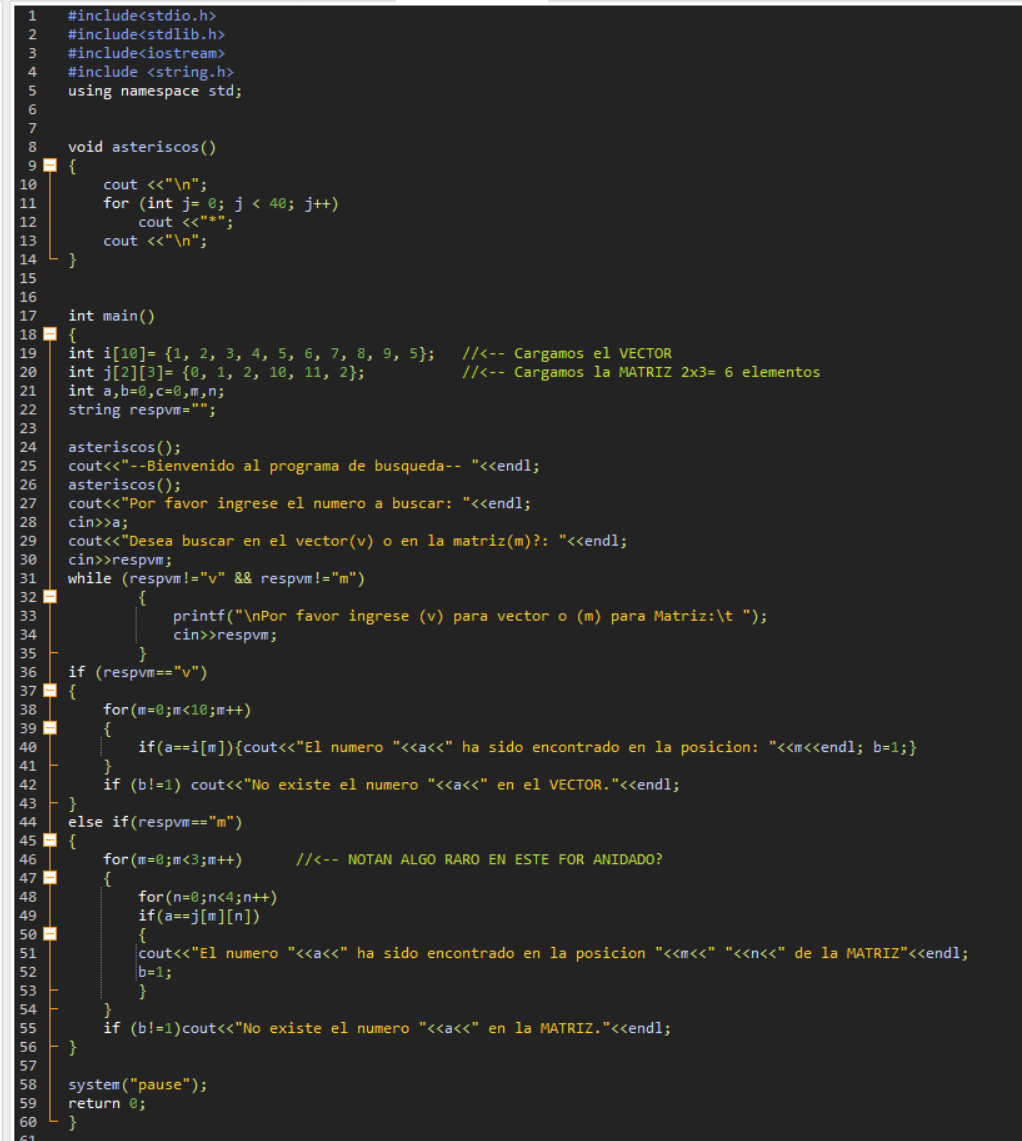


**Ejemplo Matriz 2:**

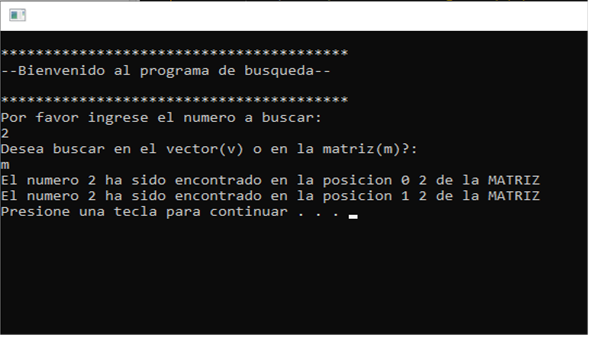
         Vamos a utilizar 1 matriz y 1 vector que ya han sido inicializadas, es decir que ya contienen un valor asignado en sus parámetros.

int i[10]= {1, 2, 3, 4, 5, 6, 7, 8, 9, 5};

int j[2][3]= {0, 1, 2, 10, 11, 2};



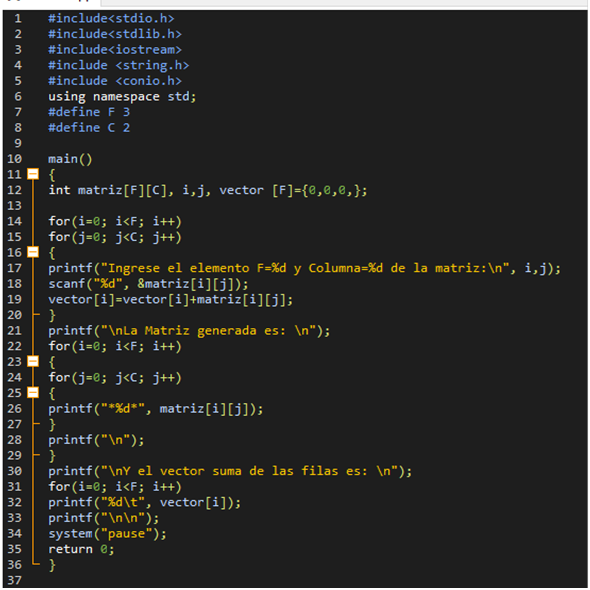
     Como podemos observar este programa permite buscar un valor que haya sido pre-cargado dentro de una matriz y un vector denominados i[10] y j[2][3], funcionalmente es bastante fácil de comprender, vamos a utilizar una función void asteriscos para presentar el sistema y luego ingresaremos el número a buscar con un for y finalmente seleccionaremos si vamos a buscar en el vector o la matriz.



     En la pantalla del ejemplo se ha seleccionado el valor 2 y que la búsqueda se realice en la matriz. Como se puede observar al comienzo del ejercicio el valor 2 se encuentra en dos posiciones [0 2] y [1 2] es decir en la tercera y sexta posición de memoria.

**Ejemplo Matriz 3:**

     A continuación diseñaremos un programa que permita ingresar valores a una matriz de 3\*2 y luego determine la suma de cada una de las filas y la almacene en un vector.



     Los datos que debemos tener en cuenta son: utilizaremos 2 #define para los valores F y C que compondrán los valores de la matriz y de el vector, recuerden que un vector es el equivalente a una fila de la matriz, de esta forma si yo quisiera modificar la cantidad de valores de la matriz por ejemplo generar una de 6\*3 lo único que debemos hacer es modificar los valores de F y C para que todo el programa quede modificado y sea funcional. Otro dato es que voy a iniciar el vector que contendrá la suma vector [F]={0,0,0,}; esto es lo equivalente a iniciar una variable con un valor neutro. Por ultimo noten en la línea de código 18 como se realiza la asignación del valor a la matriz, no es necesario utilizar una variable sino directamente a la dirección de memoria “matriz[i][j]” le voy a ingresar el valor registrado por teclado, esto se debe a que puntualmente por ejemplo matriz[0][2] es en si una variable de memoria. Si observamos la línea de código siguiente 19 algo similar pasa con la asignación del valor de la suma al vector[i].

     Con esto damos por finalizada la clase espero puedan probar los ejercicios y ver los resultados que se obtienen de los mismos. Tengan ustedes una buena programación esta semana.