Taller sobre apuntadores - Fundamentos de programación

En clase se mencionó el concepto de *apuntador* , una variable que guarda una dirección en la memoria principal del computador. Este tipo especial de datos tiene utilidad en lenguajes de programación con tipado de datos como C. Muchas operaciones con tipo de datos complejos se realizan utilizando este mecanismo, incluyendo la implementación de arreglos en una o varias dimensiones.

Los arreglos de una dimension, como ya se mencionó, están implementados mediante un apuntador a una posición de memoria al tipo de dato de los elementos del arreglo. Por eso se puede acceder el primer elemento de un arreglo de la siguiente manera:

```
#include <stdio.h>

int main()
{
    int A[100]; // declaración de un vector de 100 posiciones de tipo
entero.
    A[0] = 100; // se inicializa el valor de la primera posición de A
    int * B; // declaración de un apuntador a entero.
    B = A; // Se iguala B a A, haciendo que B apunte a la misma dirección
en la que está guardado el inicio de A.
    printf("%d %d\n", A[0], *B); // Se imprime la primera posición del
arreglo A y luego el contenido de posición de memoria a la que apunta B
    printf("%d %d\n", A[0], *A); // Se imprime la primera posición del
arreglo A y luego el contenido de posición de memoria a la que apunta A
    // en el segundo printf nos aprovechamos directamente del hecho que el
arreglo A es implementador como un apuntador, por lo tanto podemos usarlo
como tal.
}
```

En el código anterior podemos ver como la implementación de los arreglos como apuntadores nos permite directamente usar la variable A, el arreglo, como un apuntador: la consecuencia de esto es que los dos printf producen el mismo resultado.

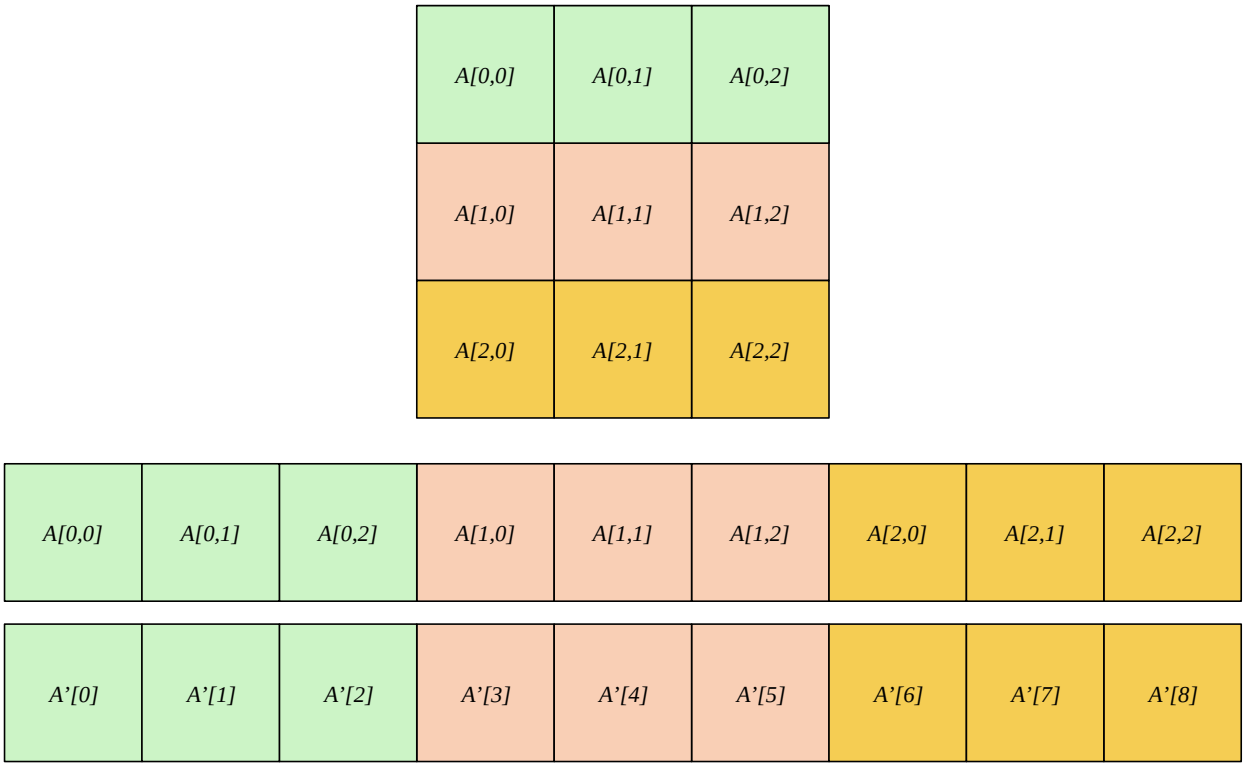
Los apuntadores se usan también para el manejo dinámico de memoria, es decir, apartar espacios de memoria en tiempo de ejecución. Para este proceso se utilizan funciones como malloc y memcpy; con estas funciones podemos crear espacios de memoria de tamaño dado por nosotros y asociarlos a apuntadores, y también, copiar el contenido de dichos espacios desde el binario.

La primera actividad a realizar es leer el siguiente material:

- How to Use Pointers in C Programming
- La sección sobre apuntadores en C - C Pointers (con sus dos subsecciones)

- La sección sobre manejo de memoria en C - C Memory Management (completa, las cinco subsecciones)
- Relación entre apuntadores y arreglos

Otra consideración importante es la que a veces hay que realizar a la hora de trabajar con arreglos de n dimensiones, es decir, con matrices. El mecanismo nativo de C involucra el uso de *apuntadores a apuntadores* : cada fila de la matriz está representada por un apuntador al primer elemento de la fila, y los primeros elementos de cada fila se agrupan en un arreglo de apuntadores. Por lo tanto, la matriz resultante es representada por un apuntador al primer elemento de un arreglo de apuntadores. Esto suena algo complejo, y puede llegar a serlo, generando problemas en programas que requieran matrices con una gran cantidad de elementos. Una alternativa a esto es utilizar arreglos de una dimension organizando los elementos de una manera llamada orden de fila principal. Utilizando esta representación podemos guardar todos los elementos de una matriz de n filas por m columnas en un arreglo unidimensional de nxm elementos, lidiando solo con un apuntador. Podemos ver esta representación en la siguiente ilustración:



Como aclaración, las filas se representan con el primer indice, y las columnas con el segundo: de esta manera $ A[i, j] $ representa el elemento de A en la fila $ i $ y la columna $ j $. En la ilustración se ve una matriz de 3 filas por 3 columnas y su correspondiente representación en orden de fila principal. cada para elemento en i, j puede referenciarse mediante un indice único en la transformación. A continuación podemos ver las equivalencias en forma de tabla:

A[i,j]	***A'[n_i]***
A[0,0]	*A'[0]*

A[i,j]	*A'[n_i]*
A[0,1]	*A'[1]*
A[0,2]	*A'[2]*
A[1,0]	*A'[3]*
A[1,1]	*A'[4]*
A[1,2]	*A'[5]*
A[2,0]	*A'[6]*
A[2,1]	*A'[7]*
A[2,2]	*A'[8]*

La transformación de la pareja de indices $ i, j $ a indice único $ n_i $ se hace mediante la formula $ n_i = i * maxcols + j $ donde $ maxcols $ es el valor de columnas de la matriz. No entra dentro del alcance de este taller, pero la transformación se puede extender a matrices de 3 o más dimensiones mediante una formula generalizada; se puede consultar más al respecto en varios recursos, por ejemplo, aquí (Esto es algo más avanzado, así que se puede dejar para después)

Con este ordenamiento, el realizar operaciones con matrices se simplifica, sobretodo a la hora de usar funciones, especialmente funciones con parámetros por referencia. Consultar más sobre funciones en C en los recursos siguientes:

- Funciones en C
- Funciones en C - C Functions (las cuatro subsecciones)

La actividad principal de este taller es construir un programa que realice reducción de matrices aumentadas de n x n + 1 en C, basándose en el procedimiento explicado en clase para Python, pero usando manejo dinámico de memoria. El programa también deberá calcular la multiplicación de dos matrices, que para efectos de este taller puede ser calcular la multiplicación de una matriz dada por ella misma: Es decir, el procedimiento debe ser general como para cualquier par de matrices, pero para probarlo pueden multiplicar una matriz por ella misma, esto con el fin de reducir código de lectura de datos de la matriz.

Para los diversos procedimientos es posible que requieran de la función memcpy, pueden consultar información de referencia sobre esta es varios recursos, por ejemplo Función memcpy

El resultado de este taller será un código fuente en C que realice las operaciones necesarias para reducir la matriz aumentada de n x n + 1 es decir:

- Leer el valor de filas y de columnas, n.
- Generar la matrix n x n + 1 en memoria mediante el uso de apuntadores. Construir varias matrices similares para guardar versiones iniciales de la matriz así como para salvar resultados intermedios (recordar la función memcpy).
- Leer los datos de la matriz n x n + 1, y almacenarla en la matriz creada en memoria, usando la representación de orden de fila principal.
- Realizar la reducción de la matriz, basándose en la realizada en Python (cabe recordar que deben reemplazar las operaciones de corte de lista que Python hace automáticamente por ciclos anidados).

- Realizar la multiplicación de matrices usando la definición estándar de multiplicación de matrices.