Laboratorio 3

Introducción a punteros

El objetivo del presente laborario es brindar una introducción al tema punteros y arreglos multidimensionales

Duración del proyecto: 1 clase.

Para compilar los archivos de este laboratorio, podes elaborar a mano un Makefile como se vio en clases.

Ejercicio 1: arreglos multidimensionales

Primera parte: Completar la carga de datos

Abrí el archivo ../input/weather_cordoba.in para ver cómo vienen los datos climáticos.
Cada línea contiene las mediciones realizadas en un día.
Las primeras 3 columnas corresponden al año, mes y día de las mediciones. Las restantes 6 columnas son la temperatura media, la máxima, la mínima, la presión atmosférica, la humedad y las precipitaciones medidas ese día.
Para evitar los números reales, los grados están expresados en décimas de grados (por ejemplo, 15.2 grados está representado por 152 (décimas)). La presión también ha sido multiplicada por 10 y las precipitaciones por 100, o sea que están expresadas en centésimas de milímetro). Esto permite representar todos los datos con números enteros. Vos no necesitás multiplicar ni dividir estos valores, esta información es sólo para que puedas entender los datos.

Lo primero que tenés que hacer es completar el procedimiento de carga de datos en el archivo array_helpers.c. Además de observar el resto del archivo array_helpers.c para entenderlo, podés revisar el mismo archivo de la primera semana del lab. Tenes que completar donde dice "Complete aquí".

Una vez que lo hayas hecho podés testear si la carga funciona correctamente ejecutando:

$ gcc -Wall -Werror -Wextra -pedantic -std=c99 -c array_helpers.c weather.c

*$ gcc -Wall -Werror -Wextra -pedantic -std=c99 -o weather *.o main.c*

$ ./weather ../input/weather_cordoba.in > weather_cordoba.out

Con el flag -Werror no compilará (ver explicación en ejercicio 2). Remové el flag “Werror” y compilá. Ejecutá el código como esta. Que podes observar en la salida del programa?

Si no hubo ningún error, ahora podés comparar la entrada con la salida:

$ diff ../input/weather_cordoba.in weather_cordoba.out

Si esto último no arroja ninguna diferencia, significa que tu carga funciona correctamente.

Parte 2: Análisis de lo datos

Crear los archivos:

- weather_utils.c: Implementación de las funciones mencionadas a continuación.
- weather_utils.h: Declaración de las funciones a exportar.

Modificar el archivo main.c para que se muestren los resultados de todas las funciones:

1. Implementar una función que obtenga la menor temperatura mínima histórica registrada en la ciudad de Córdoba según los datos del arreglo.
2. Implementar un procedimiento que registre para cada año entre 1980 y 2016 la mayor temperatura máxima registrada durante ese año.

   Ayuda: El procedimiento debe tomar como parámetro un arreglo que almacenará los resultados obtenidos.

   ```
   void procedimiento(tclimate(a), int output[YEARS]) {
       ...
       for (unsigned int year = 0; year < YEARS; year++) {
           ...
           output[year] = ... // la mayor temperatura máxima del año  'year' + 1980
       }
   ```

3. Implementar un procedimiento que registre para cada año entre 1980 y 2016 el mes de ese año en que se registró la mayor cantidad mensual de precipitaciones.

Ejercicio 2: Ordenación de un arreglo de tuplas

Abrí el archivo ../input/atpplayers.in para visualizar los datos. Es un listado por orden alfabético

de jugadores profesionales de tenis. El nombre del jugador viene acompañado de una abreviatura de su pais, el número que ocupa en el ranking, su edad, su puntaje y el número de torneos jugados en el último año.

También podés observar todos los cambios que debieron hacerse en las descripciones de las funciones de sort.c y helpers.c (las que no son funciones nuevas) para adaptarlo al nuevo tipo de arreglo (en lugar de arreglos de enteros de antes).

Como está ahora si compilas con:

$ gcc -Wall -Werror -Wextra -pedantic -std=c99 -c helpers.c sort.c

El código no compilará. ¿Porque?. Si bien aparecen algunos errores por pantalla, vemos que estos son en realidad *warnings* del compilador. El compilador "advierte" que hay situaciones en el código que podrán llevar a errores de codificación. En este caso en particular dichos errores **SABEMOS** que corresponden a que hay funciones sin terminar, entonces **SOLO** en este caso podemos desactivar el flag "Werror" que hace que los *warnings* de compilación sean tratados como errores.

Entonces, hasta que terminemos de implementar los algoritmos de sorting:

$ gcc -Wall -Wextra -pedantic -std=c99 -c helpers.c sort.c

*$ gcc -Wall -Wextra -pedantic -std=c99 -o mysort *.o main.c*

$ ./mysort ../input/atpplayers.in

Ahora el código compila, no es cierto?

RECORDATORIO: No olvides volver a poner la flag al momento de compilar finalmente tu código. Tené en cuenta que al momento del parcial los profes compilaremos usando el flag "Werror" y si no compila el código con esta flag habilitada te restará puntos.

Ahora podes comprobar que la salida es idéntica a la entrada (salvo por la información sobre

el tiempo utilizado para ordenar). Para comprobar eso, hacé

$ ./mysort ../input/atpplayers.in > atpplayers.out

$ diff ../input/atpplayers.in atpplayers.out

El ejercicio es realizar los cambios que necesites en el archivo sort.c para ordenar

el arreglo cargado de modo de que el listado de salida sea en el orden según su ranking. Podes reutilizar el código del laboratorio anterior realizando las modificaciones que creas pertinentes y utilizar aquí cualquiera de los algoritmos de ordenación vistos en clase: insertion sort, selection sort. quick sort, etc.

Ejercicio 3: Uso básico de punteros.

El objetivo de este ejercicio es adquirir un entrenamiento básico y comprender el funcionamiento de punteros en C.

Un puntero es una variable especial que guarda una dirección de memoria.

Representamos a los punteros usando '*'. Es decir que int* es una variable de tipo puntero a int.

Para el manejo de punteros contamos con dos operadores unarios básicos

- Dereferenciación (*): obtiene el **valor** de lo apuntado por el puntero. Supongamos que tenemos una variable de tipo int* llamemoslá p. *p retornará el valor entero que se aloja en dirección de memoria p.
- Referenciación (&): obtiene la dirección de memoria de una variable. Supongamos que tenemos un entero 'x'. int x; &x retornará la dirección de memoria que aloja a la variable x.

Para pensar:

- Que valor tendrá 'y' luego de ejecutar el siguiente código:

  int x = 3;

  int y = 10;

  *y = *(&x);*

- Recordas en el lab 1 cuando hablamos de funciones como fscanf? ¿que parámetros tomaba dicha función?

La tarea consiste en completar el archivo "main.c" de manera que la salida del programa por pantalla sea la siguiente:

x = 9

m = (100, F)

a[1] = 42

Las restricciones son:

- No usar las variables 'x', 'm' y 'a' en la parte izquierda de alguna asignación.
- Se pueden agregar líneas de código, pero no modificar las que ya existen.
- Se pueden declarar hasta 2 punteros.

Recuerda siempre inicializar los punteros en NULL:

```
int *p = NULL;
```

NULL es una macro definida en los headers de C como ‘0’ que se utiliza para representar al puntero que no apunta a ningún lugar, también llamado nulo.

Ayuda: La clase pasada vimos como hacer debugging de un programa con GDB. Esta herramienta también es útil para entender “que esta pasando” con mi código cuando se ejecuta. Intentá compilar con símbolos de debugging y poner breakpoints para imprimir los valores de las variables. También podes imprimir valores como:

- Tamaño de una variable: print sizeof(x)
- Dirección de memoria de una variable: print &x
- El valor que hay en la memoria apuntada por un puntero: print *p

Ejercicio 4: Aprovechando punteros para hacer el código más eficiente

La intención del ejercicio es explorar la conveniencia de utilizar punteros para

que los intercambios (swaps) sean más eficientes.

Completar el archivo “sort.c” copiando código del ejercicio 2 y realizando las modificaciones pertinentes para trabajar con arreglos de punteros a tuplas.

Notar que la función 'main' muestra la cantidad de tiempo empleado en la ordenación.

¿Funciona más rápido la versión con punteros? ¿Por qué son más eficientes los intercambios con esta versión?

La última línea de la función main antes del return llama a “destroy” ¿porque? ¿Que ocurriría si esa línea no estuviera ahí?