

Algoritmos y Estructuras de Datos II 2020

Laboratorio 2: Arreglos multidimensionales, estructuras y punteros

Ejercicio 1: Arreglos multidimensionales

En la carpeta ejer1 disponés de los siguientes archivos:

- **array_helpers.h** : contiene descripciones de funciones para cargar y volcar al disco los datos climáticos.
- **array_helpers.c** : contiene implementaciones de dichas funciones, la de cargar está incompleta.
- **main.c** : contiene al programa principal que invoca al procedimiento de carga de los datos climáticos e inmediatamente procede a volcarlos por la salida estándar.

Parte 1: 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 "PLEASE COMPLETE".

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

Primero debes compilar ejecutando:

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

y ahora podés correr el programa, ejecutando:

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

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 los datos

1) Crear los siguientes archivos:

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

2) Declarar e implementar las siguientes funciones en weather.h y weather.c respectivamente:

- a) Implementar una función que obtenga la menor temperatura mínima histórica registrada en una ciudad de Córdoba según los datos del arreglo.
- b) 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(int a[YEARS][MONTHS][DAYS][PHYS_QTTYS], int output[YEARS]) {  
    ...  
    for (unsigned int year = 0; year < YEARS; year++) {  
        ...  
        output[year] = ... // la mayor temperatura máxima del año 'year' + 1980  
    }  
}
```

- c) 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.

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

Una vez que lo hayas hecho podés compilar 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
```

y ahora podés correr el programa, ejecutando:

```
./weather ../input/weather_cordoba.in
```

Ejercicio 2: Ordenar un arreglo de estructuras

En la carpeta ejer2 disponés de los siguientes archivos:

- **helpers.h** : contiene descripciones de funciones para cargar y volcar al disco datos de los jugadores del tenis profesional.
- **helpers.c** : contiene implementaciones de dichas funciones.
- **sort.h** : descripciones de algoritmo de ordenación.
- **sort.c** : su implementación incompleta.
- **main.c** : contiene al programa principal que por ahora invoca al procedimiento de carga de los datos y los vuelve a volcar por la salida estándar.

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 país, 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 podés compilar ejecutando:

```
gcc -Wall -Werror -Wextra -pedantic -std=c99 -c helpers.c sort.c
gcc -Wall -Werror -Wextra -pedantic -std=c99 -o mysort *.o main.c
```

y ahora podés correr el programa, ejecutando:

```
./mysort ../input/atpplayers.in
```

y 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 consiste en 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.

Una vez completados compilar ejecutando:

```
gcc -Wall -Werror -Wextra -pedantic -std=c99 -c helpers.c sort.c
gcc -Wall -Werror -Wextra -pedantic -std=c99 -o mysort *.o main.c
```

y ahora podés correr el programa, ejecutando:

```
./mysort ../input/atpplayers.in
```

Ejercicio 3: Punteros en C

Un puntero en C es una variable que contiene la **dirección de memoria** de otra variable del mismo tipo.

La sintaxis de declaración de un puntero es la siguiente:

```
data_type *pointer_name;
```

Cuando se define una variable, se asigna una **dirección de memoria** para esta variable, en el cual se guardara un valor.

El lenguaje C provee dos operadores de punteros: operador de dirección (&) y operador de indirección (*).

Si x es el nombre de variable, entonces $\&x$ será su dirección de memoria.

Si p es la variable puntero, entonces $*p$ será el valor del objeto al cual apunta p .

Parte 1) Completar el archivo `operator.c` usando los operadores de punteros $\&$ y $*$, donde dado un puntero a entero p , debes cambiar el valor referido de manera tal que la salida del programa por pantalla sea la siguiente:

$x = 60$

Tenes que completar donde dice "PLEASE COMPLETE".

Una vez completado, compilá ejecutando:

```
gcc -Wall -Werror -Wextra -pedantic -std=c99 -o operator operator.c
```

y ahora podés correr el programa, ejecutando:

```
./operator
```

Parte 2) Completar el archivo `main.c` donde dados dos punteros a enteros, p y q ,

- intercambie los valores referidos sin modificar los valores de p y q .
- intercambie los valores de los punteros.

Sea un tercer puntero a un entero r que inicialmente es igual a p ,

¿cuales serán los valores de $*p$, $*q$ y $*r$ luego de ejecutar el programa en cada uno de los dos casos?

Tenes que completar donde dice "PLEASE COMPLETE".

Una vez completado, compilá ejecutando:

```
gcc -Wall -Werror -Wextra -pedantic -std=c99 -o swap main.c
```

y ahora podés correr el programa, ejecutando:

```
./swap
```

Parte 3) Uso básico de punteros.

Completá el archivo "pointers.c" de manera tal 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;
```

Tenes que completar donde dice "PLEASE COMPLETE".

Una vez completados, compilá ejecutando:

```
gcc -Wall -Werror -Wextra -pedantic -std=c99 -o pointers pointers.c
```

y ahora podés correr el programa, ejecutando:

```
./pointers
```

