
Ejercicio integrador del lenguaje C

Punteros, arreglos, archivos.
Estructuración de sources y headers

Problemática

Definir un procedimiento que dados dos arreglos, uno de temperaturas y otro de humedades, registre los días con la temperatura más alta que 29° y una humedad mayor al 50%.

Las temperaturas y humedades se encuentran en dos archivos distintos que deben ser parametrizados por consola.

¿cómo podemos resolver esto?

Indexación vs Aritmética de punteros

En la teoría vimos que cualquier operación que se pueda lograr mediante subíndices de arreglos también se puede realizar con punteros. En general, la solución que involucre los punteros será más rápida pero quizás un poco más difícil de comprender.



Indexación vs Aritmética de punteros

En la teoría vimos que cualquier operación que se pueda lograr mediante subíndices de arreglos también se puede realizar con punteros. En general, la solución que involucre los punteros será más rápida pero quizás un poco más difícil de comprender. **Sea cual sea la convención elegida, es importante no mezclar ambas notaciones ya que puede traer problemas que requieren tiempo y realizar trazas.**

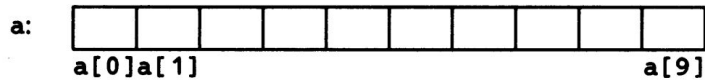


Conceptos a tener en cuenta

Supongamos que...

```
int a[10];
```

Declara un arreglo de 10 bloques consecutivos de enteros.



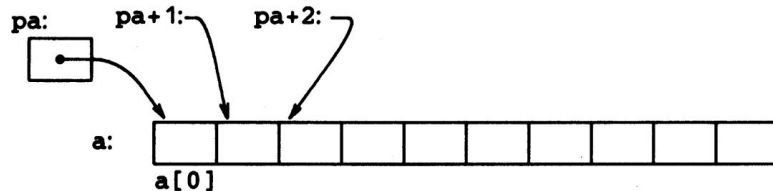
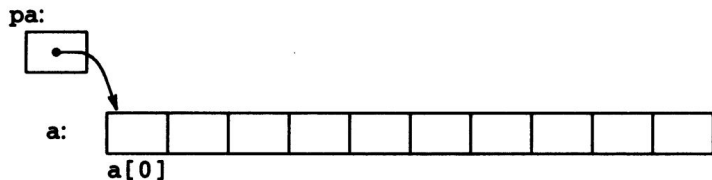
Utilizar **indexación** implica que accedo al arreglo utilizando la notación $a[i]$

```
int* pa;
```

Declara un puntero de tipo entero, por lo cual, una asignación utilizando el operador de enreferenciamiento hace que pa apunte al elemento cero de a ; es decir, pa contiene la dirección de $a[0]$

```
pa = &a[0];
```

Utilizar **aritmética de punteros** implica que $pa + i$ es la dirección de $a[i]$, y $*(pa + i)$ es el contenido de $a[i]$.



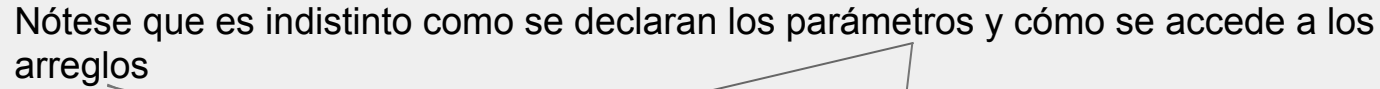
```
void dias_tropicales(int temperaturas[], int* humedades, int dias){
    int i, cant=0;

    for( i=0; i< dias; i++){
        if(cumple_propiedad(*(temperaturas+i),humedades[i]) > 0){
            cant++;
        }
    }
    printf("La cantidad de días tropicales fue: %i \n", cant);
}
```

Definición del procedimiento dias_tropicales

—

Nótese que es indistinto como se declaran los parámetros y cómo se accede a los arreglos



```
void dias_tropicales(int temperaturas[], int* humedades, int dias){
    int i, cant=0;

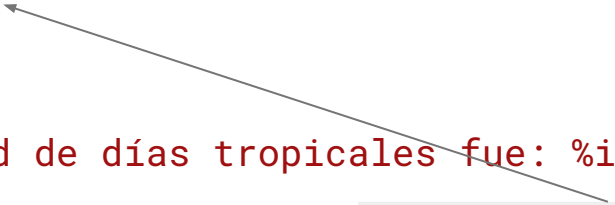
    for( i=0; i< dias; i++){
        if(cumple_propiedad(*(temperaturas+i), humedades[i]) > 0){
            cant++;
        }
    }
    printf("La cantidad de días tropicales fue: %i \n", cant);
}
```

Definición del procedimiento dias_tropicales

—

Nótese que es indistinto como se declaran los parámetros y cómo se accede a los arreglos

```
void dias_tropicales(int temperaturas[], int* humedades, int dias){  
    int i, cant=0;  
  
    for( i=0; i< dias; i++ ){  
        if(cumple_propiedad(*(temperaturas+i), humedades[i]) > 0){  
            cant++;  
        }  
    }  
    printf("La cantidad de días tropicales fue: %i \n", cant);  
}
```



```
int cumple_propiedad(int temp, int  
hum){  
    return temp>29 && hum>50;  
}
```

Definición del procedimiento dias_tropicales

Archivos

Un archivo es un conjunto de datos estructurados. Hay dos tipos de archivos: de texto y binarios.

En este caso trabajaremos con **texto**, ya que se constituyen como una **secuencia de caracteres dispuestas en líneas terminadas por un caracter de nueva línea o fin de archivo**. Se puede conseguir la entrada y salida de datos a un archivo a través del uso de la biblioteca de funciones `<stdio.h>`

Nombre	Función
fopen()	Abre un archivo.
fclose()	Cierra un archivo.
fgets()	Lee una cadena de un archivo.
fputs()	Escribe una cadena en un archivo
fseek()	Busca un byte específico de un archivo.
fprintf()	Escribe una salida con formato en el archivo.
fscanf()	Lee una entrada con formato desde el archivo.
feof()	Devuelve cierto si se llega al final del archivo.
ferror()	Devuelve cierto si se produce un error.
rewind()	Coloca el localizador de posición del archivo al principio del mismo.
remove()	Borra un archivo.
fflush()	Vacía un archivo.

Funciones brindadas por la librería stdio.h

Puntero, apertura, cierre, lectura

El tipo FILE (su estructura) permite hacer referencias a archivos. *Está declarado en el header de stdio.h*

```
FILE *archivo;
```

La función fopen() abre el nombre del archivo al que apunta por sistema de archivos utilizando el modo dado.

```
FILE *fopen(const char nombre_archivo, const char modo);
```

La función fclose() cierra un archivo que fue abierto por fopen(). Escribe toda la información que todavía se encuentra en el buffer y lo cierra a nivel sistema operativo.

```
int fclose(FILE *archivo);
```

Para leer de un archivo abierto, utilizaremos en este caso la función fscanf. Se utiliza para leer la entrada de un archivo a través de un formato definido.

```
int fscanf(FILE *stream, const char *format, ...)
```

Fuentes

- [fopen](#)
- [fclose](#)
- [fscanf](#)
- [feof](#)

```
int main(int argv, char* args[]){  
    int j=0, dias = 5;  
    int temperaturas[dias], humedades[dias];  
    FILE *fTemperatura, *fHumedad;
```

Declaración de función main y variables

```
int main(int argv, char* args[]){
    int j=0, dias = 5;
    int temperaturas[dias], humedades[dias];
    FILE *fTemperatura, *fHumedad;

    fTemperatura = fopen(args[1], "r");
    fHumedad = fopen(args[2], "r");

    if (fTemperatura == NULL || fHumedad == NULL){
        perror("Error en la apertura de archivos.");
        exit(ARCHIVO_CORRUPTO);
    }
```

Apertura de archivo, chequeo de errores

```
int main(int argv, char* args[]){
    int j=0, dias = 5;
    int temperaturas[dias], humedades[dias];
    FILE *fTemperatura, *fHumedad;

    fTemperatura = fopen(args[1], "r");
    fHumedad = fopen(args[2], "r");

    if (fTemperatura == NULL || fHumedad == NULL){
        perror("Error en la apertura de archivos.");
        exit(ARCHIVO_CORRUPTO);
    }
}
```

Cuando leemos por comando, es el arreglo de caracteres *args* quien posee los nombres de los archivos con los cuales trabajaremos.

Dichos archivos **deben encontrarse en el mismo directorio** donde se encuentra el **programa principal**, y en las posiciones 1 y 2 del arreglo encontraremos los nombres parametrizados por consola.

Apertura de archivo, chequeo de errores

```
while(!feof(fTemperatura) && !feof(fHumedad) && j<dias ){
    fscanf(fTemperatura, "%d", &temperaturas[j]);
    fscanf(fHumedad, "%d", &humedades[j]);

    printf("Temperatura %i : %i - ", j, *(temperaturas+j));
    printf("Humedad %i : %i \n", j, humedades[j]);

    j++;
}
```

Recorrido de archivo, copia de datos a arreglos locales

—

```
while(!feof(fTemperatura) && !feof(fHumedad)){
    fscanf(fTemperatura, "%d", &temperaturas[j]);
    fscanf(fHumedad, "%d", &humedades[j]);

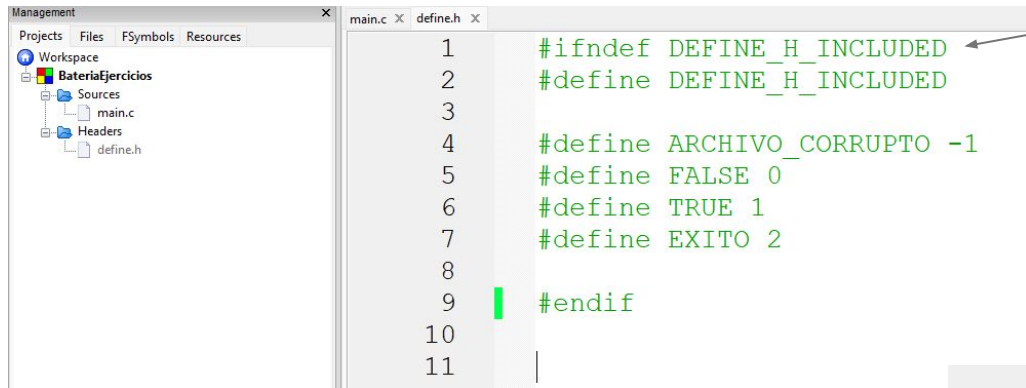
    printf("Temperatura %i : %i - ", j, *(temperaturas+j));
    printf("Humedad %i : %i \n", j, humedades[j]);

    j++;
}

dias_tropicales(temperaturas, humedades, dias);

fclose(fTemperatura);
fclose(fHumedad);
return EXITO;
}
```

Invocación a procedimiento, cierre de archivos, retorno exitoso

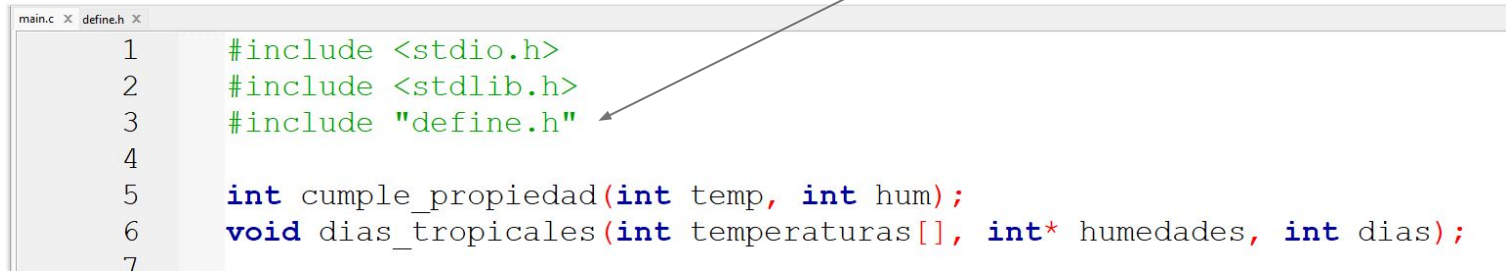


```
1  #ifndef DEFINE_H_INCLUDED
2  #define DEFINE_H_INCLUDED
3
4  #define ARCHIVO_CORRUPTO -1
5  #define FALSE 0
6  #define TRUE 1
7  #define EXITO 2
8
9  #endif
10
11
```

Definición de define.h

Declaración de macros

Inclusión de *define.h*, prototipos de funciones



```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "define.h"
4
5  int cumple_propiedad(int temp, int hum);
6  void dias_tropicales(int temperaturas[], int* humedades, int dias);
7
```

Macros, mensajes de error