Punteros

Entendiendo punteros

1. Dadas las definiciones de las variables en la siguiente porción de código ¿qué espera ver por pantalla utilizando los `printf()` a continuación? Explique la salida.

```c
float * pfloat, manzana = 40.0, pera = 35.0;

printf("&pfloat: %p\n", &pfloat);
printf("&manzana: %p\n", &manzana);
printf("&pera: %p\n", &pera);

printf("pfloat: %p\n", pfloat);

pfloat = &manzana;
printf("pfloat: %p\n", pfloat);
printf("*pfloat: %p\n", *pfloat);

pfloat = &pera;
printf("pfloat: %p\n", pfloat);
printf("*pfloat: %p\n", *pfloat);
```

2. Indicar el significado de las siguientes expresiones, cuáles son correctas, cuáles incorrectas y por qué.

```c
int i = 3, * pint;
float f = 10.0;

pint = &i;
*pint = 10;
*pint = f;
pint = &f;
pint = 4321;
```

3. Analizar las siguientes expresiones. Indicar si hay errores y justificar.

a.
```c
float var, set[] = {1.0, 2.0, 3.0, 4.0, 5.0};
float *p;

p = set;
var = *p;
*p=10.0;
```

b.
```c
int conj[5], list[]={5, 4, 3, 2, 1};
int *punt;
```

```
3
4  punt = list;
5  conj = punt;
6  list = conj;
7  punt = &conj;
```

c.

```
1  int *pint, arrayint[5];
2  float *pfloat, arrayfloat[5];
3
4  pint = arrayint;
5  pfloat = arrayfloat;
6  pint += 1;
7  pfloat += 1;
8  pint++;
9  pfloat++;
10 pint -= 1;
11 pfloat -=1;
```

d.

```
1  int *p, a[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
2
3  int var;
4  p = a;
5  *p = 33;
6  var = *p;
7  var = *(p+1);
8  var = *(p+3);
9  var = *(p++);
10 var = *p++;
11 var = *(++p);
12 var = *++p;
13 var = ++*p;
14 var = (*p)++;
15 var = ++(*p);
16 var = *p+1;
```

e.

```
int *p, a[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

var = *(a+1);
var = *(a++);
```

4. Dadas las siguientes declaraciones:

```
int *ip, **ipp, (*ip16)[16], i, j;
int matriz[5][16];
```

Indique cuál o cuáles de las siguientes expresiones son válidas, cuál o cuáles pueden producir errores y qué error puede o pueden generar:

a. `ip16 = matriz;`

b. `ip = (int *)matriz;`

c. `ipp = (int **) matriz;`

d. `*(*(ip16 + i) + j)`

e. `*(*(matriz + i ) + j)`

5. Escribir un programa que imprima cada uno de los elementos de un arreglo dos dimensional utilizando un puntero para acceder a los mismos, en lugar de utilizar subíndices. Utilizar el siguiente arreglo y los punteros indicados abajo:

```
int matriz[3][4] = { { 1, 2, 3, 4}, { 5, 6, 7, 8}, { 9,10,11,12} };
int * ptr;
int (*ptr2vector)[4];
int fila, col;
```

Funciones con punteros

1. Implementar una función `swap` que reciba 2 datos `a` y `b` por puntero y los intercambie, de forma tal que `b` sea `a` y viceversa.
2. Implementar las siguientes funciones, retornando los resultados por la interfaz:
   - `void suma(int a, int b, long int *suma)`,
   - `void resta(int a, int b, long int *resta)`,
   - `void producto(int multiplicando, int multiplicador, float *producto)`,
   - `void division(int dividendo, int divisor, double *)`, y
   - `mod()`: retorna el resto de una división.
3. Escribir una función que convierta un número que representa una cantidad de segundos, a su equivalente en horas, minutos y segundos, retornando las partes por la interfaz, `bool` por el nombre, indicando el resultado de la operación, `true` de ser posible `false` si no. Realizar las validaciones pertinentes.
4. **(matemática)** Implementar funciones que reciban un arreglo de números y su longitud y realicen las operaciones detalladas abajo. Utilice notación de punteros exclusivamente.
   a. completar el vector con ceros,
   b. completar el vector con unos,
   c. calcular y retornar la suma por la interfaz,
   d. calcular y retornar la media por la interfaz,
   e. calcular y retornar la varianza por la interfaz,
   f. retornar el valor máximo del arreglo por la interfaz,
   g. retornar el valor mínimo del arreglo por la interfaz,
   h. modificar los elementos del vector reemplazándolos por sus valores al cuadrado,

i. modificar los elementos del arreglo reemplazando cada elemento por su signo.
Considere $\text{sign}(x) = \begin{cases} -1 & \text{si } x < 0 \\ 1 & \text{si } x \geq 0 \end{cases}$

j. modificar los elementos del arreglo reemplazándolos por las diferencias finitas de primer orden. Para hacer esto, usando como caso de ejemplo el primer y segundo elemento:

```
*(v + 0) = *(v + 1) - *(v + 0);
*(v + 1) = *(v + 2) - *(v + 1);
```

5. Implementar una función que reciba 2 vectores y sus longitudes, y copie el contenido de uno en el otro, usando el siguiente prototipo y notación de punteros en la implementación, la función debe devolver `true` de poder realizar la operación `false` en caso contrario:

```
bool veccpy(double *dest, size_t ldest, const double *orig, size_t lorig);
```

6. Implementar una función que reciba 2 vectores y sus longitudes, y retorne un valor booleano, por la interfaz, si los vectores son iguales.

7. **(búsqueda lineal)** Implementar una función que reciba un vector, su longitud, un número objetivo a buscar y retorne **un puntero** a la posición en la que se encuentra. ¿Qué ocurre en caso que el objetivo no se encuentre y qué se retorna? Implemente 2 versiones de la función, utilizando los siguientes prototipos:

```
const int * busqueda_lineal(const int *v, size_t n, int objetivo);
bool busqueda_lineal(int *v, size_t n, int objetivo, int **resultado);
```

8. **(búsqueda binaria)** Implemente una función similar de búsqueda, utilizando búsqueda binaria. Prototipos:

```
const int * busqueda_binaria(const int *v,  /* vector donde buscar */
                             size_t n,      /* largo del vector */
                             int objetivo,  /* elemento a buscar */
                             size_t izq,    /* franja del vector donde
buscar, */
                             size_t der);   /* delimitada por izq y der */
bool busqueda_binaria(int *v,        /* vector donde buscar */
                      size_t n,      /* largo del vector */
                      int objetivo,  /* elemento a buscar */
                      size_t izq,    /* franja del vector donde buscar, */
                      size_t der,    /* delimitada por izq y der */
                      int **resultado);
```

Última actualización: September 3, 2022