Ejercitación – Serie N
Puntero a Función - PaF

Conceptos que se agregan en esta serie: El acceso a funciones a través de variables que poseen almacenada su dirección (variables tipo puntero a función). Array de punteros a función.
Como ya se sabe, está prohibido el uso de variables globales.
Implementar las aplicaciones, estructuras de datos y funciones complementarias que permita visualizar el resultado de las funciones realizadas.

Ejercicio 1: Imprimir cualquier array

Realizar una función que tenga la capacidad de imprimir cualquier tipo de array.
La función debe tener el siguiente prototipo:

```
void print(void *base, size_t nitems, size_t size, int (*prt)(const void *));
```

o

```
void print(void *base,  int  nitems ,  int  size , int (*prt)(const void *));
```

en donde

- **base**: puntero al primer elemento del array a imprimir.
- **nitems**: cantidad de elementos del array (apuntado por base).
- **size**: tamaño en bytes de cada elemento del array.
- **prt**: función que imprime un elemento del array.

Considerar para la prueba de funcionamiento mínimo los siguientes tipos de arrays:

- array de números enteros o de punto flotante. `int vec[MAX];`
- array de estructuras. `Struct xxx vec[MAX];`
- matriz de char utilizada como array de string. `char mat[FILAS][COLUMNAS];`
- Array de char * (array de strings). `char *vec[MAX];`

Ejercicio 2: Imprimir cualquier array en forma condicional

Agregar a la función anterior la posibilidad de incluir una condición para imprimir el dato. Por ejemplo, que sean números pares, que el string comience con mayúscula, etc.
La inclusión de este condicional debe ser tal que pueda aplicarse para cualquier tipo de array.
La función debe retornar la cantidad de elementos impresos.

Ejercicio 3: Para pensar (y hacer)

Basado en el ejercicio anterior, hacer las modificaciones necesarias para que solo se impriman los elementos que sean mayores al promedio, teniendo en cuenta que puede ser un array de cualquier tipo de dato numérico e incluso una estructura que posea un dato numérico (promediable).

Nota: *Se entiende que no se debe enviar el promedio. La función a desarrollar debe tener la capacidad de calcularlo*.

Ejercicio 4: Búsqueda sobre cualquier array

Realizar una función que tenga la capacidad de buscar un elemento dentro de un array el cual no tiene por qué estar ordenado.
La función debe tener el mismo prototipo que la función **bsearch** de la stdlib.

```
void *bsearch(const void *key, const void *base, size_t nitems, size_t size,
                                        int (*compar)(const void *, const void *));
```

Tener en cuenta que la función debe retornar el puntero al elemento encontrado o NULL en el caso de no haber coincidencia.

Tener presente que el tipo de dato de **key**, no tiene por qué ser igual al de **base**. Por ejemplo, teniendo un array de estructuras buscar por un campo interno de la estructura (por ejemplo, un long int).

Ejercicio 5: Búsqueda sobre cualquier array

En base al ejercicio anterior, considerar que la coincidencia del parámetro buscado no tenga por qué ser exacta. Un valor numérico dentro de un determinado rango (por ejemplo +/- 2%), un string que inicie o contenga parte de un texto, o la coincidencia de algún (o algunos) parámetros dentro de una estructura son solo algunos ejemplos.

Analizar cual sería la mejor manera de implementar este tipo de búsqueda. Realice un par de implementaciones a modo de ejemplo.

Ejercicio 6: Multi localización sobre cualquier array

Modificar el ejercicio anterior de manera que la función pueda retornar múltiples coincidencias.

a.- la función devuelva un array de punteros
b.- la función devuelva una lista simplemente enlazada.
c.- ¿De qué otra manera se podría retornar múltiples coincidencias? Impleméntelo.

El orden de los elementos encontrados debe ser coincidentes a como se encuentran en el array original.

Ejercicio 7:

Modificar el ejercicio anterior de manera que las múltiples coincidencias se retornen en el sentido del array original o a la inversa según se defina (en los argumentos de la función).

Ejercicio 8: Array de PaF

Realizar una aplicación que permita realizar las operaciones básicas a través de un array de puntero a función.
Tomar como base el siguiente código.

```
...
printf("ingrese 1er valor:\n");
scanf("%f", &a);
printf("ingrese 2do valor:\n");
scanf("%f", &b);
...
printf("Seleccione una operación:\n");
printf("1. Suma\n2. Resta\n3. Multiplicación\n4. División\n :");
scanf("%d", &opcion);
...
res=pFunc[opcion](a,b);
```

Ejercicio 9: Ordenamiento de cualquier array

Realizar una función que tenga la capacidad de ordenar cualquier tipo de array.
La función debe tener el mismo prototipo que la función **qsort** de la stdlib.

```
void qsort(void *base, size_t nitems, size_t size,
                                    int (*compar)(const void *, const void *));
```

Implementar la función utilizando los algoritmos de burbujeo o intercambio.

Ejercicio 10:

Mediante la función anterior (o la función **`qsort`**), ordenar un array de números enteros bajo las siguientes consideraciones:

- Primero vienen los números positivos pares en forma ascendente.
- Seguido por los números positivos impares en forma descendente.
- Luego los números negativos en forma ascendente
- Finalmente, los ceros

Ejercicio 11: Array de PaF - 2

Se posee el siguiente array de puntero a función.

```
int (*vec[100])={func_00,func_01,NULL,func_03,…,func_xx,NULL,func_yy,…,func_97,NULL,NULL};
```

Ya se sabe que si el indicé que se obtiene para acceder a la función indicada por el array está fuera de rango, esto provocará la caída de nuestra aplicación (o al menos un mal funcionamiento).

Pero ¿qué ocurre si el índice está en rango, pero el contenido es NULL?

Mmmm, ¿también se cae?.

Si. Correcto. También se cae.

Se pide entonces implementar el mecanismo para indexar el array y acceder a la función correspondiente, sin que esto pueda provocar la caída de la aplicación. Queda a su criterio como actuar en cada caso.

Ejercicio 12:

Encapsular el código anterior en una función, que retorne la dirección de la función (es decir retorne un puntero a función). La función debe recibir el índice para acceder al array con las direcciones de las funciones.

Suponer que las funciones declaradas en el array poseen el siguiente formato: `int (*) (int, void*)`