Práctica 2: *Estructuras/Uniones/Campos de bits*

Contenido:
Esta práctica está diseñada para que el estudiante comience a programar usando las estructuras de datos utilizadas más comúnmente en aplicaciones más cercanas a las que se enfrentará como ingeniero electrónico.

1) Dada la siguiente estructura que permite representar tiempos:

```
typedef struct tiempo{
   int anio, mes, dia, hora, minuto, segundo;
} Tiempo;
```

Implemente las siguientes funciones:

1. `int compara_tiempos( Tiempo *t1, Tiempo * t2);`
   que retorna:

1	si *t1 es anterior a *t2
0	si *t1 es igual a *t2
-1	si *t1 es posterior a *t2

2. `void imprime_tiempo( Tiempo t );`
   que imprime el contenido de la estructura **t** con el siguiente formato: "**dia/mes/anio hora:minuto:segundo**" (por ejemplo: "3/8/1974 18:23:59").

Para representar un archivo se escogió la siguiente estructura que utiliza la anterior:

```
typedef struct {
        char * nombre;
        Tiempo ultima_mod;
     } Archivo;
```

Donde, **nombre** es un cadena de caracteres con el nombre del archivo y **ultima_mod** es una estructura **Tiempo** que almacena la fecha y hora en que se modificó por última vez el archivo. Dado un arreglo **lista** de **n** archivos, implemente funciones para ordenarlo alfanuméricamente y temporalmente de acuerdo a los siguientes prototipos

a. **`void ordena_alfa( Archivo * lista, int n );`**
   (utilice la función **strcmp** de la biblioteca estándar para comparar los nombres de los archivos al ordenarlos alfabéticamente).

b. **`void ordena_temporal(Archivo * lista, int n );`**
   (utilice la función **compara_tiempos** para comparar los tiempos de última modificación de los archivos).

2) Dada la siguiente estructura

```
typedef struct
{
  int n;
  double *coeficiente;
} Polinomio;
```

Que representa a un polinomio de orden n, cuyos n+1 coeficientes $c_0, c_1, \ldots, c_n$, almacenados en el vector apuntado por `coeficientes`, corresponden a las potencias $x^0, x^1, \ldots, x^n$, cree las siguientes funciones:

```
a.    /* crea y retorna un nuevo Polinomio */
Polinomio * creaPolinomio(int orden);
b.    /* asigna el n_ésimo coeficiente del Polinomio */
void setCoef(int n, double c, Polinomio * P);
c.    /* retorna el n_ésimo coeficiente del Polinomio */
double getCoef(int n, Polinomio * P);
d.    /* calcula el polinomio en x usando:
((...((c[n]*x+c[n-1])*x+c[n-2])*x+ ...+c[1]*x)+c[0]) */
double especializa( double x, Polinomio * P );
e.    /* suma dos Polinomios retorna un nuevo Polinomio con el resultado
*/
Polinomio * sum( Polinomio *p1, Polinomio *p2);
f.    /* multiplica dos Polinomios y retorna un nuevo Polinomio con el
resultado */
Polinomio * mult( Polinomio *p1, Polinomio *p2);
g.    /* deriva un Polinomio retornando un nuevo Polinomio con el
resultado */
Polinomio * deriv( Polinomio *p );
h.    /* libera la memoria asociada con el polinomio */
void destruyePolinomio( Polinomio *p );
i.    /* busque por el método de bisección un cero de un polinomio dentro
      de un intervalo [a,b], con una precisión dada por épsilon y lo
      retorne*/
double ceropol( Polinomio *p, double a, double b, double epsilon);
```

3) La siguiente estructura de datos permite representar números enteros con precisión arbitraria:

```
typedef struct {
        char sign;
        unsigned char num_bytes;
        unsigned char * bytes;
    } APint;
```

Donde `sign` es el signo (-1 0 o 1), `num_bytes` es el número de bytes que se utilizan para representar el número (por ejemplo 4 equivaldría aproximadamente a un `int` en una arquitectura de 32 bits), `bytes` es un arreglo que contiene los valores con los que se representa el número.
Deberá implementar las siguientes funciones para sumar y multiplicar dichos enteros:

```
a.    APint * suma_APint( APint * i1, APint * i2);
```

b. APint * producto_APint(APint * i1, APint * i2);

4) Se define TipoCiudad como un struct para almacenar la posición de una ciudad en una representación de dos dimensiones, es decir, en un plano.

```
struct TipoPunto{
      double abscisa;
      double ordenada;};
struct TipoCiudad{
      TipoPunto situacion;
      char nombre[50];
};
```

Para almacenar varias ciudades, se construirá un vector de TipoCiudad. Se pide construir una función que, a partir de un vector de TipoCiudad y dado el nombre de una ciudad, reordene ascendentemente el vector atendiendo a la distancia euclídea del resto de las ciudades con respecto a la elegida. Por ejemplo, si elegimos "Granada", la ciudad con dicho nombre deberá ponerse como la primera componente del vector; la segunda será la ciudad más cercana a "Granada" y así sucesivamente. El prototipo de la función será:

```
void Reordenar (TipoCiudad ciudades[], int num_ciudades, const char
                nombre_ciudad_referencia[]);
```

Recordemos que la distancia euclídea entre dos puntos se define como la raíz cuadrada de la suma de los cuadrados de las diferencias de las abscisas y las ordenadas. No pueden usarse vectores auxiliares.

5) Escribir un programa para sumar dos matrices:

- Cree una estructura con dos enteros con las dimensiones y un puntero a la matriz.
- Llene con valores enteros en forma aleatoria.
- Desarrolle una función que realice la suma pasando como argumento las estructuras de matrices y devuelva el puntero al resultado.
- Mostrar el resultado de la suma en forma matricial.

```
struct matrix { short filas, col; int **matriz; };
```

6) Dadas las siguientes estructuras de datos

```
typedef struct{
      unsigned char R, G, B; //componentes primarios de un color
} RGB;

typedef struct{
  int ancho, alto;
  RGB **pixel;
} ImagenRGB;

typedef unsigned char Gris;
```

```
typedef struct{
  int ancho, alto;
  Gris **pixel;
} ImagenGris;
```

Implemente las siguientes funciones:

a. `/* crea y retorna una nueva ImagenRGB */`
   `ImagenRGB * creaImagenRGB( int ancho, int alto );`
b. `/* crea y retorna una nueva ImagenGris */`
   `ImagenGris * creaImagenGris( int ancho, int alto );`
c. `/* asigna el pixel de la fila y columna dadas */`
   `void setPixelRGB( ImagenRGB *im, int fila, int columna, RGB * pix );`
d. `/* asigna el pixel de la fila y columna dadas */`
   `void setPixelGris( ImagenGris *im, int fila, int columna, Gris * pix );`
e. `/* retorna el pixel de la fila y columna dadas */`
   `RGB * getPixelRGB( ImagenRGB *im, int fila, int columna );`
f. `/* retorna el pixel de la fila y columna dadas */`
   `Gris getPixelGris( ImagenGris *im, int fila, int columna );`
g. `/* libera la memoria asociada con la imagen im */`
   `void destruyeImagenRGB( ImagenRGB * im);`
h. `/* libera la memoria asociada con la imagen im */`
   `void destruyeImagenGris( ImagenGris * im);`
i. `/* convierte un pixel RGB en uno Gris usando la fórmula:`
   `Gris = 0.299*R+0.587*G+0.114*B */`
   `Gris RGBtoGris( RGB * pix );`
j. `/* transforma la imagenRGB en una` *nueva* `ImagenGris */`
   `ImagenGris * transforma( ImagenRGB * im );`

7) Crear un programa que defina los siguientes tipos de variables:

```
short int          intCor = 29813;          // Entero corto 16 bits.
int                intEst = 548524874;   //Entero estandar 32 bits. También es equivalente "long int"
long long int      intLar = -84467073709551615;        // Entero largo 64 bits
float              decEst = 4.7575f;       // Decimal punto flotante estandar 32 bits
double             decLar = -7.348744198498e2f;        // Decimal punto flotanto largo 64 bits
unsigned long long int        uintLart = 18446744073709551610; // Entero largo 64 bits sin signo
```

Descomponer cada variable en bytes utilizando uniones, ídem usando punteros.
Verificar su valor y mapa de bits.
Analizar el orden de almacenamiento de cada byte.
Reasignar valores usando hexadecimales y verificar como se almacenan.

8) Cree un campo de bits (representando un byte bit a bit) y una unión que contenga dicho campo, además de otro que permita representar el código ASCII de un caracter, de forma de poder utilizarlos en una función que, recibiendo como argumento una cadena de caracteres, cambie en cada carácter de la misma, los bits que ocupan las posiciones pares (cambie de 0 a 1 o de 1 a 0). La función deberá retornar la cadena de caracteres modificada. Por ejemplo si la cadena de caracteres pasada como argumento contiene un carácter 'a' (código ASCII=97, en binario: 01100001), conmutando los bits 0, 2, 4 y 6, en binario tengo 00110100 o, en decimal: 52 que corresponde a '4'.

9) Una red de sensores inalámbrica intercambia mensajes entre sus nodos y esta comunicación tiene diferentes características. La comunicación puede ser broadcast o unicast, multihop o single hop, el mensaje se puede enviar o puede fallar su envío, y el mismo puede ser recibido o no. Escriba una función en C, que utilice un campo de bits, para representar que tipo de características tiene la comunicación y si se pudo establecer o no. Realice un programa que genere 6 valores al azar de un byte que sirva para representar un tipo particular de comunicación y analice que tipo de comunicación se estableció (atención: no sirve cualquier número ya que, cada bit en una posición par del mismo (posición par), debe tener un valor opuesto al siguiente bit (el que está en posición par+1)). Muestre por pantalla como resultó cada comunicación.

Significado de cada bit: 0 broadcast, 1 unicast, 2 multihop, 3 single hop, 4 enviado, 5 no enviado, 6 recibido y 7 no recibido.

10) Construir una estructura de datos que represente a un automóvil con los siguientes atributos:

- numeroDominio: número de patente,
- numeroMotor: número que trae grabado el motor,
- marca: marca del automóvil
- modelo: modelo de automóvil
- tamañoMotor: potencia del motor.
- color: color de la carrocería

Escribir las siguientes funciones para manipular un automóvil:

- crearAuto: genera una instancia de tipo Automóvil con valores de inicialización adecuados para cada uno de sus atributos.
- cambiarColor: modifica el valor del atributo color de un automóvil que recibe
- mostrarAuto: imprime todos los atributos de un automóvil en particular

Escribir un programa que permita al usuario generar una lista de automóviles (usando estructuras autoreferenciadas), los muestre ordenados según el número de dominio, permita modificar cualquiera de los atributos de un automóvil, ofrezca la posibilidad de agregar un nuevo automóvil a la lista, eliminar un automóvil de la lista, permita determinar cuál es el automóvil de mayor potencia presente en la lista.

11) Un dispositivo de comunicaciones puede utilizar una velocidad de transferencia de 1200, 2400, 4800, y 9600 baudios. Definir un tipo enumerativo que modelice dicha situación.

Referencias:

Algunos ejercicios fueron extraídos de:

1. http://www.ib.cnea.gov.ar/~servos/CursoC/
2. http://materias.fi.uba.ar/7502E/material.html