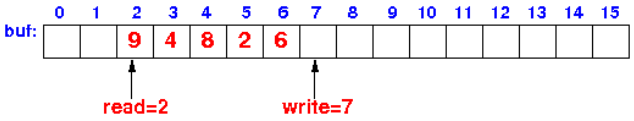
Circular Buffers

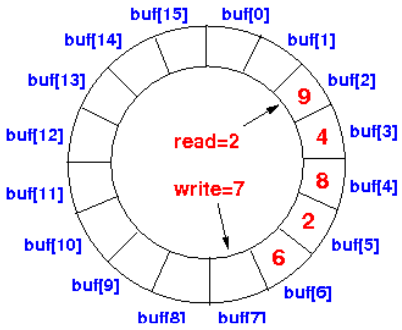
Mandatory: (part of CasioCAN project)

Un método muy efectivo de pasar información del puerto serial a una aplicación es mediante buffer circulares. Mientras que la interrupción por puerto serial está escribiendo en el buffer, la propia aplicación se encargará de leer el contenido del buffer. Esto permite a la aplicación no necesitar estar al pendiente de forma constante a la información que va llegando al puerto serial.

El control de la escritura y lectura se realiza por medio de un puntero o variable llamados head (o write) y tail (o read), éstos incrementan su valor cada que se escribe o se lee y a su vez detectan el final del buffer para volver a comenzar desde el primer índice

Esquema de un buffer circular





1.- Crear un driver que configure y controle un buffer circular, el driver se escribira en los archivos **buffer.h** y **buffer.c**. Ya es hora que aprendas como compilar un programa con mas de un archivo Programs with more than one file

2.- El buffer circular debera manejar **n** elementos de tipo **uint8_t**.

3.- Debera tener una interfaz que incilize el buffer

```
void Buffer_Init( Buffer_t *hbuffer );
```

Inicializa el buffer circular colando los elementos head y tail a cero, y los valores de empty a uno y full a cero.

4.- Debera teneruna interfaz que permita escribir en el buffer

```
void Buffer_Write( Buffer_t *hbuffer, unsigned char data );
```

Escribe un nuevo dato de 8 bits en el buffer si hay espacio disponible, de no haberlo no se escribirá dato alguno

5.- Debera tener una interfaz que permita leer el buffer

```
unsigned char Buffer_Read( Buffer_t *hbuffer );
```

Lee un dato del buffer, el dato que es leído ya no existirá dentro del buffer. Si el buffer está vacío no se leerá ningún dato, y el valor regresado por la función no será válido

6.- Debera tener una interfaz que informe si el buffer esta vacio

```
unsigned char Buffer_IsEmpty( Buffer_t *hbuffer );
```

La función regresa un uno si no hay más elementos que se puedan leer del buffer circular y un cero si al menos existe un elemento que se pueda leer. Es necesario usar esta función antes de usar la función de lectura.

7.- La estructura de control del buffer deberá tener los siguientes elementos:

```
typedef struct
{
    unsigned char   *Buffer;
    unsigned long  Elements;
    unsigned long  Head;
    unsigned long  Tail;
    unsigned char   Empty;
    unsigned char   Full;
    //agregar más elementos si se requieren
} Buffer_t;
```

8.- Los elementos **Buffer** y **Elements** deben inicializarse antes de la llamar la función de inicilizacion.

9.- ejemplo:

```
//inicialización
unsigned char arreglo[200];
Buffer_t  CircBuffer;

:
:
CircBuffer.Buffer = arreglo;
CircBuffer.Elements = 200u;
Buffer_Init( &CircBuffer );

:
:
//leemos todos los mensajes del buffer
whie( Buffer_IsEmpty( &CircBuffer )==0u )
{
    data = Buffer_Read( &CircBuffer );
    :
    :
}

```

```
:
:
//escribimos en el buffer
Buffer_Write( &CircBuffer, rx_dato );
```