

Constantes, definición estructuras, etc. necesarias en varios .c :

- Declarar en un .h
- Todos los .c que las necesitan incluyen a ese .h

Variables globales necesarias en varios .c :

- Uno de ellos las declara “normal”, y si lo desea las inicializa:

uno.c	
	int _modo_control=MODO_CONTROL_REF; ...

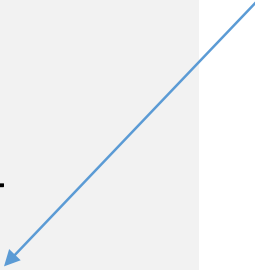
- El resto las declara “extern”, y no las puede inicializar:

otro.c	
	extern int _modo_control; ...

Esquema solución ampliación 1:

1.1) main() se quedará a espera de conexiones TCP -> se necesita interrupción para teclado

```
char _cmd[80]="";  
void KeyboardISR()  
{  
    char new_char=_RX_SerialComm;  
    Según new_char: (ojo, mantener carácter nulo terminando siempre cadena)  
        '\r' o '\n' : fin de línea ->  
            ProcesarCadena(_cmd); -> Comando a variables  
            GuardarEnLog(_cmd);  
            Vaciar(_cmd);  
        '\b' : (tecla Del) -> quitar último carácter de _cmd, borrar de pantalla  
        otro: añadir al final de _cmd, escribir en pantalla  
}  
  
printf("\b\b\b");
```



\b = back space, retrasa el curso

Esquema solución ampliación 1:

1.2) main()

```
SOCKET _connected; // Global: Necesario en main y en función de interrupción temporiz
main()
{
    ... otras variables locales ...
    SOCKET server; // Solo necesario en main()
    struct sockaddr_in dir_local, dir_remoto;
    char rcv_sock[80];

    Inicializar sockets con WSASStartup()
    Crear socket server con llamada a socket()
    Hacer bind() para el socket server en dirección 0.0.0.0 (cualquiera) y puerto 44444
    Ejecutar listen() para el socket server
    _connected=INVALID_SOCKET;

    ... otras inicializaciones (incluida interrupción para teclado) ...

    Bucle while espera conexiones y comunica (apartado siguiente)

}
```

Esquema solución ampliación 1:

1.3) Bucle comunicación en main() : recepción de comandos si control IP activado

```
while (1)
{
    _connected=accept(server,...); // Esperar solicitud de conexión (crea nuevo socket para
    if (! _control_ip_activado) // Lo decide ProcesaComando()
    {
        closesocket(_connected); // No se aceptan conexiones
        _connected=INVALID_SOCKET;
    }
    while (_connected!=INVALID_SOCKET)
    {
        Leer con recv(_connected,...) en vble rcv_sock
        Si no error lectura -> Procesa rcv_sock con ProcesaComando()
        Si error lectura (fallo comu) -> closesocket(_connected) y poner a INVALID_SOCKET
    }
}
```

Esquema solución ampliación 1:

1.4) Envío de datos en interrupción temporizada si control ip activado

```
void ISR_Control( )
{
    .... Un paso del lazo de control ...
    if ( _connected!=INVALID_SOCKET)
    {
        char snd_sock[80];
        Componer cadena snd_sock con sprintf()
        Enviar con send(_connected,snd_sock,....)
    }
}
```

Esquema solución ampliación 2:

- 2.1) Variables globales tipo WINDOW* para cada una de las ventanas (se usan en main y en la función de interrupción)

```
WINDOW *_ventana_cmd, *_ventana_estado, ....;
```

- 2.2) Inicializar curses y ventanas en main()

```
main()
{
    ...
    Llamadas a funciones para iniciar curses

    // Creacion de una ventana
    _ventana_cmd=newwin(.....);
    watttron(_ventana_cmd, colores para la ventana);
    resto de inicialización de la ventana (borrar, poner marco, etc.)
    wrefresh(_ventana_cmd);

    ... Resto de inicializaciones
}
```

Esquema solución ampliación 2:

2.3) Escribir en una ventana cuando sea necesario

```
...  
    wmove(id ventana, posición deseada del cursor en la ventana);  
    wclrtoeol(id ventana); // Si se desea borrar hasta fin de línea  
    wprintw(id ventana, resto como en printf);  
...  
    wrefresh(id ventana);
```

2.4) Esperar por cadena en una ventana (si no se usa interrupción teclado)

```
...  
    wmove(id ventana, posición deseada del cursor en la ventana);  
    wclrtoeol(id ventana); // Si se desea borrar hasta fin de línea  
    wprintw(id ventana, resto como en printf);  
    wrefresh(id ventana);  
    wgetstr(id ventana, cadena de caracteres);
```

Esquema solución ampliación 3:

2.1) Derivación de la posición para obtener la velocidad

$$Vel_k = (pos_k - pos_{k-1}) / T_m \quad (\text{Ojo unidades, necesaria tabla para } pos_k)$$

2.2.) Integración trapezoidal de la velocidad para obtener la posición

$$pos_k = pos_{k-1} + (vel_k + vel_{k-1}) / 2 * T_m \quad (\text{Ojo unidades, necesarias tablas para } pos_k \text{ y } vel_k)$$

iii Atención !!! : la integración va acumulando errores, es normal que con el paso del tiempo pos_k se aleje de la realidad

Esquema solución ampliación 4:

2.1) Variables globales necesarias

entero _modo_limpia -> TRUE/FALSE

Tabla 2 enteros _estado_SW -> para detectar cambios en los switches

Tabla 3 enteros _ultimos_pulsadores_activados -> indica los 3 últimos switches activados

2.2) En función de interrupción:

```
DesplazaTablaInt(_estado_SW,2);
_estado_SW[0]=_RB_Puerto1;
if (se detecta un cambio en un pulsador) // _estado_SW[0] distinto de _estado_SW[1]
{
    DesplazaTablaInt(_ultimos_pulsadores_activados,3);    // Sólo se desplaza si hay
                                                         // cambio en algún pulsado

    Si se ha activado SW5 (antes inactivo, ahora activo) ->
        _ultimos_pulsadores_activados[0]=5; // El cambio más reciente es la activación
                                           // del pulsador 5

    else if (id. para SW6)
    else if (id. para SW7)
    else (Nada de lo anterior) ->
        _ultimos_pulsadores_activados[0]=-1; // El más reciente cambio no nos vale
```

Esquema solución ampliación 4:

2.2) Continuación

```
if (secuencia correcta _últimos_pulsadores_activados)
{
    Modo control: posición con referencia teclado, valor ref teclado = 70
    _limpia_activado=TRUE;
}

if (_limpia_activado)
{
    if (desactivados todos SW5 y SW6 y SW7) -> _limpia_activado=FALSE;

    if (posk[0] > 69.5)
        ref teclado = -70

    if (posk[0] < -69.5)
        ref teclado = 70
}
.... Resto del lazo de control, el control de posición hará el trabajo ....
}
```