



- 282 Removals + 122 Additions

```

2 #include <string.h>
3 #include <stdlib.h>
4 #include <stdio.h>
5 #include <unistd.h>
6 #include <sys/types.h>
7 #include <fcntl.h>
8 #include <time.h>
9 #include <stdlib.h>
10 #include <stdio.h>
11 #include <unistd.h>
12 #include <sys/types.h>
13 #include <sys/time.h>
14 #include <termios.h>
15 #include <fcntl.h>
16 #include <string.h>
17 #include <errno.h>
18 #include <syslog.h>
19 #include <ctype.h>
20 #include <pthread.h>
21 #include <unistd.h>
22 #include <sys/mount.h>
23 #include <sys/types.h>
24 #include <signal.h>
25 #include "serial.h"
26 #include "../comun/aes.cpp"
27 #ifdef INET
28 #include "../comunicacion/servidor.cpp"
29 // #include "cliente.c"
30 #endif
31 #include "../comun/Configuracion.h"
32 #include "../comun/Funciones.cpp"
33 #include "../limitador/Lectura2C.cpp"
34 #include "../comun/Configurador.cpp"
35 #include "../comun/EstadoDelLimitador.h"
36 #include "../limitador/Registrador.cc"
37 #include "../limitador/AtenuadorPga.cpp"
38 #define FormatoFecha "%d/%d/%d-%d:%d"
39 Funciones funciones;
40 int puerto;
41 #define NombreDelPuerto "/dev/ttyS0"
42 bool continua;
43 bool identificado = true;
44 bool ParaConexionRemota()
45 {
46 #ifdef INET
47 return true;
48 #endif
49 return false;
50 }
51 void post()
52 {
53 /*
54 puts("PrePost");
55 system("chmod -R 777 /var/slr 2>/dev/null >/dev/null");
56 puts("PostOk");
57 */
58 }
59 char *fecha(time_t f)
60 {
61 static char linea[30];

```

```

2 #include <string.h>
3 #include <stdlib.h>
4 #include <stdio.h>
5 #include <unistd.h>
6 #include <sys/types.h>
7 #include <fcntl.h>
8 #include <time.h>
9 #include <stdlib.h>
10 #include <stdio.h>
11 #include <unistd.h>
12 #include <sys/types.h>
13 #include <sys/time.h>
14 #include <termios.h>
15 #include <fcntl.h>
16 #include <string.h>
17 #include <errno.h>
18 #include <syslog.h>
19 #include <ctype.h>
20 #include <pthread.h>
21 #include <unistd.h>
22 #include <sys/mount.h>
23 #include <sys/types.h>
24 #include <signal.h>
25 #include "serial.h"
26 #ifdef INET
27 #include "../comunicacion/servidor.cpp"
28 // #include "cliente.c"
29 #endif
30 #include "../comun/Configuracion.h"
31 #include "../comun/Funciones.cpp"
32 #include "../comun/Configurador.cpp"
33 #include "../comun/EstadoDelLimitador.h"
34 #include "../limitador/Registrador.cc"
35 #define FormatoFecha "%d/%d/%d-%d:%d"
36 Funciones funciones;
37 int puerto;
38 #define NombreDelPuerto "/dev/ttyS0"
39 bool continua;
40 bool identificado = true;
41 bool ParaConexionRemota()
42 {
43 #ifdef INET
44 return true;
45 #endif
46 return false;
47 }
48 void post()
49 {
50 /*
51 puts("PrePost");
52 system("chmod -R 777 /var/slr 2>/dev/null >/dev/null");
53 puts("PostOk");
54 */
55 }
56 char *fecha(time_t f)
57 {
58 static char linea[30];

```

```

62  strftime(linea, 17, "%02d/%02m/%4Y %02H:%02M",
    localtime(&f));
63  return linea;
64  }
65  void escribe(char *cadena)
66  {
67  int longitud = strlen(cadena);
68  write(puerto, cadena, longitud);
69  }
70  void printSID()
71  {
72  char *c = _hfiSujkSQ;
73  char salida[1024];
74  bzero(salida, 1024);
75  for (int i = 0; i < strlen(c); i += 3)
76  {
77  salida[i / 3] = c[i];
78  }
79  escribe(salida);
80  }
81  void printSerial()
82  {
83  Configuracion config;
84  char salida[1024];
85  bzero(salida, 1024);
86  snprintf(salida, 1024, "%d", config.numeroDeSe
    rie);
87  escribe(salida);
88  }
89  char *leeCadena(char *texto = "")
90  {
91  escribe(texto);
92  static char cadena[350];
93  bzero(cadena, 350);
94  char c = 'a';
95  for (int i = 0; i < 350; i++)
96  {
97  read(puerto, &c, 1);
98  if (!isalnum(c) &&
99  (c != '/') && (c != ' ') &&
100 (c != '.') && (c != ':') &&
101 (c != 'ñ') && (c != 'Ñ') &&
102 (c != 'a') && (c != 'Á') &&
103 (c != 'é') && (c != 'É') &&
104 (c != 'í') && (c != 'Í') &&
105 (c != 'ó') && (c != 'Ó') &&
106 (c != 'ú') && (c != 'Ú') &&
107 (c != ',') && (c != ';') &&
108 (c != '_') && (c != '-') &&
109 (c != 'o') && (c != 'a') &&
110 (c != '=') && (c != '*'))
111 break;
112 cadena[i] = c;
113 }
114 //printf("cadena Leida=<<%s>>\n",cadena);
115 return cadena;
116 }
117 ////////////////////////////////////////// Func
    iones de clave
118 bool claveDelLimitador()
119 {
120 Configuracion config;
121 char cadena[120];
122 snprintf(cadena, 120, "Codigo %d\n", config.nu
    meroDeSerie);
123 escribe(cadena);
124 char *respuesta = leeCadena();

```

```

59  strftime(linea, 17, "%02d/%02m/%4Y %02H:%02M",
    localtime(&f));
60  return linea;
61  }
62  void escribe(char *cadena)
63  {
64  int longitud = strlen(cadena);
65  write(puerto, cadena, longitud);
66  }
67  void printSID()
68  {
69  char *c = _hfiSujkSQ;
70  char salida[1024];
71  bzero(salida, 1024);
72  for (int i = 0; i < strlen(c); i += 3)
73  {
74  salida[i / 3] = c[i];
75  }
76  escribe(salida);
77  }
78  void printSerial()
79  {
80  Configuracion config;
81  char salida[1024];
82  bzero(salida, 1024);
83  snprintf(salida, 1024, "%d", config.numeroDeSe
    rie);
84  escribe(salida);
85  }
86  char *leeCadena(char *texto = "")
87  {
88  escribe(texto);
89  static char cadena[350];
90  bzero(cadena, 350);
91  char c = 'a';
92  for (int i = 0; i < 350; i++)
93  {
94  read(puerto, &c, 1);
95  if (!isalnum(c) &&
96  (c != '/') && (c != ' ') &&
97  (c != '.') && (c != ':') &&
98  (c != 'ñ') && (c != 'Ñ') &&
99  (c != 'a') && (c != 'Á') &&
100 (c != 'é') && (c != 'É') &&
101 (c != 'í') && (c != 'Í') &&
102 (c != 'ó') && (c != 'Ó') &&
103 (c != 'ú') && (c != 'Ú') &&
104 (c != ',') && (c != ';') &&
105 (c != '_') && (c != '-') &&
106 (c != 'o') && (c != 'a') &&
107 (c != '=') && (c != '*'))
108 break;
109 cadena[i] = c;
110 }
111 //printf("cadena Leida=<<%s>>\n",cadena);
112 return cadena;
113 }
114 ////////////////////////////////////////// Func
    iones de clave

```

<pre> 125 return (atoi(respuesta) == config.numeroDeSerie); 126 } 127 bool claveGenerada(int add = 0) 128 { 129 srand(time(NULL)); 130 int aleatorio = (int)(((float)rand() / (float)     RAND_MAX) * 100000); 131 Configuracion config; 132 char cadena[120]; 133 snprintf(cadena, 120, "Clave %d\n", aleatorio); 134 escribe(cadena); 135 funciones.generador(aleatorio + add, cadena); 136 //puts(cadena); 137 char *respuesta = leeCadena(); 138 return (strcmp(respuesta, cadena) == 0); 139 } 140 bool password() 141 { 142 Configuracion config; 143 escribe("Password\n"); 144 char *respuesta = leeCadena(); 145 return (strcmp(respuesta, config.password) ==     0); 146 } 147 ////////////////////////////////////// Petic     ion de datos 148 time_t fechaHora(char *titulo) 149 { 150 escribe(titulo); 151 char *respuesta = leeCadena(); 152 int dia, mes, anho, horas, minutos; 153 sscanf(respuesta, "%d/%d/%d-%d:%d", &amp;dia, &amp;me     s, &amp;anho, &amp;horas, &amp;minutos); 154 struct tm td; 155 td.tm_min = minutos; 156 td.tm_hour = horas; 157 td.tm_mday = dia; 158 td.tm_mon = mes - 1; 159 td.tm_year = (anho - 1900 &lt; 0) ? anho + 100 :     anho - 1900; 160 return mktime(&amp;td); 161 }; 162 float pideNumero(char *titulo) 163 { 164 escribe(titulo); 165 return atof(leeCadena()); 166 } 167 bool confirmacion(char *titulo) 168 { 169 escribe(titulo); 170 escribe(" (s/n) : "); 171 char *respuesta = leeCadena(); 172 return (strchr(respuesta, 's')    strchr(respu     esta, 'S')); 173 } 174 bool abrePuertoSerie() 175 { 176 char *dispositivo = new char[strlen(NombreDelP     uerto) + 1]; 177 strcpy(dispositivo, NombreDelPuerto); 178 int speed = B19200; 179 struct termios portset; 180 int fd = open(dispositivo, O_RDWR   O_NOCTTY); 181 if (fd &lt; 0) 182 { </pre>	<pre> 115 bool claveGenerada(int add = 0) 116 { 117 srand(time(NULL)); 118 int aleatorio = (int)(((float)rand() / (float)     RAND_MAX) * 100000); 119 Configuracion config; 120 char cadena[120]; 121 snprintf(cadena, 120, "Clave %d\n", aleatori     o); 122 escribe(cadena); 123 funciones.generador(aleatorio + add, cadena); 124 //puts(cadena); 125 char *respuesta = leeCadena(); 126 return (strcmp(respuesta, cadena) == 0); 127 } 128 bool password() 129 { 130 Configuracion config; 131 escribe("Password\n"); 132 char *respuesta = leeCadena(); 133 return (strcmp(respuesta, config.password) ==     0); 134 } 135 ////////////////////////////////////// Petic     ion de datos 136 time_t fechaHora(char *titulo) 137 { 138 escribe(titulo); 139 char *respuesta = leeCadena(); 140 int dia, mes, anho, horas, minutos; 141 sscanf(respuesta, "%d/%d/%d-%d:%d", &amp;dia, &amp;me     s, &amp;anho, &amp;horas, &amp;minutos); 142 struct tm td; 143 td.tm_min = minutos; 144 td.tm_hour = horas; 145 td.tm_mday = dia; 146 td.tm_mon = mes - 1; 147 td.tm_year = (anho - 1900 &lt; 0) ? anho + 100 :     anho - 1900; 148 return mktime(&amp;td); 149 }; 150 float pideNumero(char *titulo) 151 { 152 escribe(titulo); 153 return atof(leeCadena()); 154 } 155 bool confirmacion(char *titulo) 156 { 157 escribe(titulo); 158 escribe(" (s/n) : "); 159 char *respuesta = leeCadena(); 160 return (strchr(respuesta, 's')    strchr(respu     esta, 'S')); 161 } 162 bool abrePuertoSerie() 163 { 164 char *dispositivo = new char[strlen(NombreDelP     uerto) + 1]; 165 strcpy(dispositivo, NombreDelPuerto); 166 int speed = B19200; 167 struct termios portset; 168 int fd = open(dispositivo, O_RDWR   O_NOCTTY); 169 if (fd &lt; 0) 170 { </pre>
--	---

```

183 return false;
184 }
185 tcgetattr(fd, &portset);
186 // We use RAW mode
187 #ifdef HAVE_CFMAKERAW
188 // The easy way
189 cfmakeraw(&portset);
190 #else
191 // The hard way
192 portset.c_iflag &= ~(IGNBRK | BRKINT | PARMRK
    | ISTRIP | INLCR | IGNCR | ICRNL | IXON);
193 portset.c_oflag &= ~OPOST;
194 portset.c_lflag &= ~(ECHO | ECHONL | ICANON |
    ISIG | IEXTEN);
195 portset.c_cflag &= ~(CSIZE | PARENB | CRTSCT
    S);
196 portset.c_cflag |= CS8 | CREAD | CLOCAL;
197 #endif
198 cfsetospeed(&portset, speed);
199 cfsetispeed(&portset, speed);
200 tcsetattr(fd, TCSANOW, &portset);
201 puerto = fd;
202 return true;
203 };
204 int abreStdin()
205 {
206     puerto = 1;
207 }
208 void muestraConfiguracion()
209 {
210     Configurador configurador;
211     FILE *f = fdopen(puerto, "w");
212     if (f == NULL)
213     {
214         escribe("No se puede mostrar configuracion\n");
215         return;
216     }
217     configurador.escribeConfiguracion(f);
218     delete (f);
219 }
220 time_t obtenFechaHora(char *titulo)
221 {
222     escribe(titulo);
223     int dia = (int)pideNumero("Dia\n");
224     int mes = (int)pideNumero("Mes\n");
225     int anho = (int)pideNumero("Año\n");
226     int horas = (int)pideNumero("Horas(24)\n");
227     int minutos = (int)pideNumero("Minutos\n");
228     struct tm td;
229     td.tm_min = minutos;
230     td.tm_hour = horas;
231     td.tm_mday = dia;
232     td.tm_mon = mes - 1;
233     td.tm_year = (anho - 1900 < 0) ? anho + 100 :
        anho - 1900;
234     return mktime(&td);
235 }
236 void test2(bool calibrarLineas = false, bool s
    inConfirmar = false)
237 {
238     int pid;
239     float mediaIzq = 0;
240     float mediaDer = 0;
241     float mediaMic = 0;
242     int ctr = 0;
243     int ganancia = 0;

```

```

171 return false;
172 }
173 tcgetattr(fd, &portset);
174 // We use RAW mode
175 #ifdef HAVE_CFMAKERAW
176 // The easy way
177 cfmakeraw(&portset);
178 #else
179 // The hard way
180 portset.c_iflag &= ~(IGNBRK | BRKINT | PARMRK
    | ISTRIP | INLCR | IGNCR | ICRNL | IXON);
181 portset.c_oflag &= ~OPOST;
182 portset.c_lflag &= ~(ECHO | ECHONL | ICANON |
    ISIG | IEXTEN);
183 portset.c_cflag &= ~(CSIZE | PARENB | CRTSCT
    S);
184 portset.c_cflag |= CS8 | CREAD | CLOCAL;
185 #endif
186 cfsetospeed(&portset, speed);
187 cfsetispeed(&portset, speed);
188 tcsetattr(fd, TCSANOW, &portset);
189 puerto = fd;
190 return true;
191 };
192 int abreStdin()
193 {
194     puerto = 1;
195 }
196 void muestraConfiguracion()
197 {
198     escribe("Configuracion: No está disponible!
        \n");
199 }
200 time_t obtenFechaHora(char *titulo)
201 {
202     escribe(titulo);
203     int dia = (int)pideNumero("Dia\n");
204     int mes = (int)pideNumero("Mes\n");
205     int anho = (int)pideNumero("Año\n");
206     int horas = (int)pideNumero("Horas(24)\n");
207     int minutos = (int)pideNumero("Minutos\n");
208     struct tm td;
209     td.tm_min = minutos;
210     td.tm_hour = horas;
211     td.tm_mday = dia;
212     td.tm_mon = mes - 1;
213     td.tm_year = (anho - 1900 < 0) ? anho + 100 :
        anho - 1900;
214     return mktime(&td);
215 }
216 void test2(bool calibrarLineas = false, bool s
    inConfirmar = false)
217 {
218     int ctr = 0;
219     Spectrum avgMic, avgLeft, avgRight;

```

<pre> 244 char *cadena = ""; 245 char *confirmacion = "";  246 if (calibrarLineas &amp;&amp; !sinConfirmar) 247 { 248 puts("Pulse 'OK' para empezar el calibrado"); 249 puts("En otro caso se saldrá"); 250 puts("Asegúrese de tener el sistema funcionand o"); 251 confirmacion = leeCadena(); 252 if (strcmp(confirmacion, "OK") != 0) 253 return; 254 } 255 // if ( (pid=fork())==0 ) 256 { 257 char cadena[120]; 258 float mic; 259 float dBmic; 260 float li; 261 float dBli; 262 float ld; 263 float dBld; 264 continua = true; 265 Lectura2C lecturas[3]; 266 Calibracion cMic; 267 Calibracion cLIz; 268 Calibracion cLDe; 269 cMic.leeCalibracion(0); 270 cLIz.leeCalibracion(1); 271 cLDe.leeCalibracion(2); 272 while (continua) 273 { 274 sleep(1); 275 lecturas[0].read(0); 276 lecturas[1].read(1); 277 lecturas[0].aplicaCalibracion(cMic, cMic); 278 lecturas[1].aplicaCalibracion(cLIz, cLDe); 279 puts("Micrófono"); 280 lecturas[0].print(); 281 puts("Líneas"); 282 lecturas[1].print(); 283 //Para volver a ponerlo como estaba usar 1Khz [5], en lugar del global. 284 dBmic = lecturas[0].globalIzquierdoPonderado (); 285 dBli = lecturas[1].globalIzquierdoPonderado(); 286 dBld = lecturas[1].globalDerechoPonderado(); 287 mic = cMic.calculaEnergia(dBmic); 288 li = cLIz.calculaEnergia(dBli); 289 ld = cLDe.calculaEnergia(dBld); 290 ctr++; 291 mediaIzq += li; 292 mediaDer += ld; 293 mediaMic += dBmic;  294 snprintf(cadena, 120, "MIC : %08.1f %3.1fdB \t", mic, dBmic); 295 //snprintf(cadena,120,"MIC : %3.1fdB\t", sono </pre>	<pre> 220 EstadoDelLimitador status; 221 char *confirmacion = ""; 222 Calibracion cal; 223 cal.leeCalibracion(1); 224 printf("Line 1\n"); 225 cal.print(); 226 cal.leeCalibracion(2); 227 printf("Line 2\n"); 228 cal.print(); 229 if (calibrarLineas &amp;&amp; !sinConfirmar) 230 { 231 puts("Pulse 'OK' para empezar el calibrado"); 232 puts("En otro caso se saldrá"); 233 puts("Asegurese de tener el sistema funcionand o"); 234 confirmacion = leeCadena(); 235 if (strcmp(confirmacion, "OK") != 0) 236 return; 237 } 238 while (true) 239 { 240 sleep(1); 241 status.actualiza(); //Si el sistema no está fu ncionando sale 242 avgMic.sum(status.mic); 243 avgLeft.sum(status.left); 244 avgRight.sum(status.right); 245 printf(" . ");  246 ctr++; 247 status.mic.show(); 248 status.left.showStereo(status.right); 249 printf("Mic : %8.1f dBA\tLeft : %8.1f dBA\tRig ht : %8.1f\n", status.mic.globalAWeighted, sta tus.left.globalAWeighted, status.right.globalA Weighted); </pre>
---	--

```

metro.derivacion(mic));
296 escribe(cadena);
297 //snprintf(cadena,120,"Liz : %08.1f    %3.1fdB
\t",li, sonometro.derivacion(li));
298 snprintf(cadena, 120, "Liz : %3.1fdB\t", dBL
i);
299 escribe(cadena);
300 //snprintf(cadena,120,"LDe : %08.1f    %3.1fdB
\n",ld, sonometro.derivacion(ld));
301 snprintf(cadena, 120, "LDe : %3.1fdB\n", dBL
d);
302 escribe(cadena);
303 if (ctr >= 10)
304 {
305 break;
306 }
307 }
308 continua = true;
309 if (calibrarLineas)
310 {
311 //puts("Lineas de entrada calibradas");
312 if (ctr < 5)
313 {
314 puts("Debe haber más de 5 muestras para la cal
ibración");
315 return;
316 }
317 mediaMic /= ctr;
318 EstadoDelLimitador estado;
319 //El sistema informa al reves
320 if (!estado.informes & IMicro)
321 {
322 puts("El micofono no esa conectado. Se usara e
l maximo");
323 Configuracion configuracion;
324 configuracion.carga();
325 mediaMic = configuracion.maximoEnEmision(time
(NULL));
326 }
327 mediaDer /= ctr;
328 mediaIzq /= ctr;
329 Calibracion cal;
330 //Guardar calibracion izquierda
331 cal.leeCalibracion(1);
332 cal.dBRef = mediaMic;
333 cal.ref = mediaIzq;
334 cal.ruido = 0;
335 cal.guardaCalibracion(1);
336 //Guardar calibracion derecha
337 cal.leeCalibracion(2);
338 cal.dBRef = mediaMic;
339 cal.ref = mediaDer;
340 cal.ruido = 0;
341 cal.guardaCalibracion(2);
342 }
343 }
344 }

250 if (ctr >= 10)
251 {
252 break;
253 }
254 }
255 continua = true;
256 if (calibrarLineas)
257 {
258 puts("Lineas de entrada calibradas");
259 if (ctr < 5)
260 {
261 puts("Debe haber más de 5 muestras para la cal
ibración");
262 return;
263 }
264 avgMic.divide(ctr);
265 avgLeft.divide(ctr);
266 avgRight.divide(ctr);
267 //Guardar calibracion izquierda
268 cal.dBRef = avgMic.dB[Spectrum_1Khz_BandInde
x];
269 cal.ref = avgLeft.energy[Spectrum_1Khz_BandInd
ex];
270 cal.ruido = 0;
271 for (int i = 0; i < Spectrum_BandCount; i++)
272 cal.equalization[i] = 0; //avgMic.dB[i]-avgLef
t[i]
273 cal.guardaCalibracion(1);
274 cal.print();
275 //Guardar calibracion derecha
276 cal.dBRef = avgMic.dB[Spectrum_1Khz_BandInde
x];
277 cal.ref = avgRight.energy[Spectrum_1Khz_BandIn
dex];
278 cal.ruido = 0;
279 for (int i = 0; i < Spectrum_BandCount; i++)
280 cal.equalization[i] = 0;
281 cal.guardaCalibracion(2);
282 cal.print();
283 }
284 }

```

```

345 void calibracion(int numero = 0)
346 {
347     if (numero == -1)
348     {
349         numero = pideNumero("Numero de sonometro :");
350         if (numero < 0 || numero > 2)
351         {
352             escribe("El numero de sonometro debe estar entre 0 y 2\n");
353             return;
354         }
355     }
356     Calibracion cal;
357     cal.leeCalibracion(numero);
358     char cadena[50];
359     sprintf(cadena, "Calibracion de sonometro %d\n", numero);
360     escribe(cadena);
361     sprintf(cadena, "Decibelios = %.1f\n", cal.dBR);
362     escribe(cadena);
363     sprintf(cadena, "Referencia = %.1f\n", cal.ref);
364     escribe(cadena);
365     sprintf(cadena, "Ruido = %.1f\n", cal.ruido);
366     escribe(cadena);
367     sprintf(cadena, "Ganancia = %d\n", cal.ganancia);
368     escribe(cadena);
369 }
370 void calibra()
371 {
372     if (identificado)
373     {
374         int numero = pideNumero("Numero de sonometro :");
375         if (numero < 0 || numero > 2)
376         {
377             escribe("El numero de sonometro debe estar entre 0 y 2\n");
378             return;
379         }
380         calibracion(numero);
381         float db = pideNumero("dBelios :");
382         float ref = pideNumero("Referencia :");
383         float ruido = pideNumero("Ruido :");
384         float ganancia = pideNumero("Ganancia :");
385     {
386         Calibracion cal;
387         cal.leeCalibracion(numero);
388         cal.ref = ref;
389         cal.dBRef = db;
390         cal.ruido = ruido;
391         cal.ganancia = (int)ganancia;
392         cal.guardaCalibracion(numero);
393         post();
394     }
395 }
396 else
397     (escribe("Debe identificarse\n"));
398 }
399 void prepara()
400 {
401     char *respuesta;
402     Configuracion configuracion;
403     if (!identificado)
404     {

```

```

285 void calibracion(int numero = 0)
286 {
287     if (numero == -1)
288     {
289         numero = pideNumero("Numero de sonometro :");
290         if (numero < 0 || numero > 2)
291         {
292             escribe("El numero de sonometro debe estar entre 0 y 2\n");
293             return;
294         }
295     }
296     Calibracion cal;
297     cal.leeCalibracion(numero);
298     char cadena[50];
299     sprintf(cadena, "Calibracion de sonometro %d\n", numero);
300     escribe(cadena);
301     sprintf(cadena, "Decibelios = %.1f\n", cal.dBR);
302     escribe(cadena);
303     sprintf(cadena, "Referencia = %.1f\n", cal.ref);
304     escribe(cadena);
305     sprintf(cadena, "Ruido = %.1f\n", cal.ruido);
306     escribe(cadena);
307     sprintf(cadena, "Ganancia = %d\n", cal.ganancia);
308     escribe(cadena);
309 }
310 void calibra()
311 {
312     if (identificado)
313     {
314         int numero = pideNumero("Numero de sonometro :");
315         if (numero < 0 || numero > 2)
316         {
317             escribe("El numero de sonometro debe estar entre 0 y 2\n");
318             return;
319         }
320         calibracion(numero);
321         float db = pideNumero("dBelios :");
322         float ref = pideNumero("Referencia :");
323         float ruido = pideNumero("Ruido :");
324     {
325         Calibracion cal;
326         cal.ref = ref;
327         cal.dBRef = db;
328         cal.ruido = ruido;
329         cal.guardaCalibracion(numero);
330         post();
331     }
332 }
333 else
334     (escribe("Debe identificarse\n"));
335 }
336 void prepara()
337 {
338     char *respuesta;
339     Configuracion configuracion;
340     if (!identificado)
341     {

```

<pre> 405 escribe("Debe identificarse\n"); 406 return; 407 } 408 muestraConfiguracion(); 409 if (confirmacion("Borrar configuracion?")) 410 { 411 unlink(F_Config); 412 } 413 if (confirmacion("Inicializar registro?")) 414 { 415 printf("Iniciando %s\n", FicheroDeRegistr o); 416 Registrador rg; 417 rg.inicializa(); 418 time_t hh = rg.horaBase(); 419 printf("Hora base fijada en %s", ctime(&amp;hh)); 420 } 421 if (!confirmacion("Desea cambiar algo? ")) 422 return; 423 configuracion.carga(); 424 configuracion.numeroDeSerie = (int)pideNumero ("Numero de Serie :"); 425 configuracion.esFrecuencial = ((int)pideNumero ("Frecuencial (&lt;0 No, &gt;0 Si: ") &gt; 0); 426 configuracion.atenuacionMaxima = (int)pideNume ro("Atenuacion Maxima: "); 427 printf("Atenuacion maximoa =%d\n", configuraci on.atenuacionMaxima); 428 printf("Numero de serie entrado %d\n", configu racion.numeroDeSerie); 429 configuracion.horaReprogramado = time(NULL); 430 respuesta = leeCadena("Local :"); 431 strcpy(configuracion.local, respuesta); 432 respuesta = leeCadena("Direccion :"); 433 strcpy(configuracion.direccion, respuesta); 434 respuesta = leeCadena("Telefono :"); 435 strcpy(configuracion.telefono, respuesta); 436 respuesta = leeCadena("Persona :"); 437 strcpy(configuracion.persona, respuesta); 438 respuesta = leeCadena("Ayuntamiento :"); 439 strcpy(configuracion.ayuntamiento, respuesta); 440 if (confirmacion("Guardar? ")) 441 { 442 puts("Grabando"); 443 configuracion.graba(); 444 puts("/Grabando"); 445 } 446 post(); 447 } 448 void EscribeIdentificacion() 449 { 450 escribe("ID:"); 451 printSerial(); 452 escribe(":"); 453 printSID(); 454 escribe("\n"); 455 } 456 void datos() 457 { 458 Configuracion configuracion; 459 configuracion.carga(); 460 if (strlen(configuracion.local) &gt; 1) 461 { 462 if (!identificado) 463 { 464 EscribeIdentificacion(); 465 if (!claveGenerada(7)) </pre>	<pre> 342 escribe("Debe identificarse\n"); 343 return; 344 } 345 muestraConfiguracion(); 346 if (confirmacion("Borrar configuracion?")) 347 unlink(F_Config); 348 if (confirmacion("Inicializar registro?")) 349 { 350 printf("Iniciando %s\n", FicheroDeRegistr o); 351 Registrador rg; 352 rg.inicializa(); 353 time_t hh = rg.horaBase(); 354 printf("Hora base fijada en %s", ctime(&amp;hh)); 355 } 356 if (!confirmacion("Desea cambiar algo? ")) 357 return; 358 configuracion.carga(); 359 strcpy(configuracion.numeroDeSerie, leeCadena ("Numero de Serie :")); 360 respuesta = leeCadena("Distribuidor :"); 361 strcpy(configuracion.distribuidor, respuesta); 362 configuracion.horaReprogramado = time(NULL); 363 respuesta = leeCadena("Local :"); 364 strcpy(configuracion.local, respuesta); 365 respuesta = leeCadena("Direccion :"); 366 strcpy(configuracion.direccion, respuesta); 367 respuesta = leeCadena("Telefono :"); 368 strcpy(configuracion.telefono, respuesta); 369 respuesta = leeCadena("Persona :"); 370 strcpy(configuracion.persona, respuesta); 371 respuesta = leeCadena("Ayuntamiento :"); 372 strcpy(configuracion.ayuntamiento, respuesta); 373 if (confirmacion("Guardar? ")) 374 { 375 puts("Grabando"); 376 configuracion.graba(); 377 puts("/Grabando"); 378 } 379 post(); 380 } 381 void EscribeIdentificacion() 382 { 383 escribe("ID:"); 384 printSerial(); 385 escribe(":"); 386 printSID(); 387 escribe("\n"); 388 } 389 void datos() 390 { 391 Configuracion configuracion; 392 configuracion.carga(); 393 if (strlen(configuracion.local) &gt; 1) 394 { 395 if (!identificado) 396 { 397 EscribeIdentificacion(); 398 if (!claveGenerada(7)) </pre>
--	--



```

466 return;
467 }
468 }
469 char *respuesta;
470 configuracion.horaReprogramado = time(NULL);
471 respuesta = leeCadena("Local :");
472 strcpy(configuracion.local, respuesta);
473 respuesta = leeCadena("Direccion :");
474 strcpy(configuracion.direccion, respuesta);
475 respuesta = leeCadena("Telefono :");
476 strcpy(configuracion.telefono, respuesta);
477 respuesta = leeCadena("Persona :");
478 strcpy(configuracion.persona, respuesta);
479 respuesta = leeCadena("Ayuntamiento :");
480 strcpy(configuracion.ayuntamiento, respuesta);
481 if (confirmacion("Guardar? "))
482 configuracion.graba();
483 post();
484 }
485 void operativo()
486 {
487 char *respuesta;
488 Configuracion configuracion;
489 if (!identificado)
490 {
491 escribe("Debe identificarse\n");
492 return;
493 }
494 muestraConfiguracion();
495 if (!confirmacion("Desea cambiar algo?"))
496 return;
497 configuracion.nDistribuidor = (long)pideNumero
("Número de Distribuidor:");
498 respuesta = leeCadena("Distribuidor :");
499 strcpy(configuracion.distribuidor, respuesta);
500 respuesta = leeCadena("Servidor :");
501 strcpy(configuracion.servidorDeDatos, respuest
a);
502 configuracion.puertoDeConexion = pideNumero("P
uerto :");
503 configuracion.minimo = pideNumero("Minimo :");
504 configuracion.maximoDeMicrofono = pideNumero
("Maximo :");
505 configuracion.medidasPorCiclo = (int)pideNumer
o("Medidas por ciclo :");
506 configuracion.tipoDeControl = (int)pideNumero
("Tipo de control (0: Mic 1: Linea 2: Mixto ):
");
507 if (configuracion.tipoDeControl == 2)
508 {
509 configuracion.rangoDeControlMixto = pideNumero
(" Margen superior referente al microfono :
");
510 configuracion.minimoDeControlMixto = pideNumer
o(" Margen inferior referente al micrófono:
");
511 }
512 if (confirmacion("Guardar?"))
513 configuracion.graba();
514 post();
515 };
516 void identifica()
517 {
518 identificado = (claveGenerada() && claveGenera
da());
519 if (!identificado)
520 escribe("Clave incorrecta\n");

```

```

399 return;
400 }
401 }
402 char *respuesta;
403 respuesta = leeCadena("Local :");
404 strcpy(configuracion.local, respuesta);
405 respuesta = leeCadena("Direccion :");
406 strcpy(configuracion.direccion, respuesta);
407 respuesta = leeCadena("Telefono :");
408 strcpy(configuracion.telefono, respuesta);
409 respuesta = leeCadena("Persona :");
410 strcpy(configuracion.persona, respuesta);
411 respuesta = leeCadena("Ayuntamiento :");
412 strcpy(configuracion.ayuntamiento, respuesta);
413 if (confirmacion("Guardar? "))
414 configuracion.graba();
415 post();
416 }
417 void identifica()
418 {
419 identificado = (claveGenerada() && claveGenera
da());
420 if (!identificado)
421 escribe("Clave incorrecta\n");

```

```

521 }
522 void configura()
523 {
524 if (!identificado)
525 {
526 escribe("Debe identificarse\n");
527 return;
528 }
529 muestraConfiguracion();
530 if (!confirmacion("Desea cambiar algo?"))
531 return;
532 Configuracion configuracion;
533 escribe("Horario DIURNO. (24h)\n");
534 configuracion.normativa.intervalo.horaDeInicio
= (int)pideNumero("\tHora de inicio :");
535 configuracion.normativa.intervalo.horaDeFin =
(int)pideNumero("\tHora de fin :");
536 configuracion.normativa.maximoDiurno = pideNum
ero("Maximo diruno :");
537 configuracion.normativa.maximoNocturno = pideN
umero("Maximo nocturno :");
538 configuracion.pasos = (int)pideNumero("Porcent
aje de atenuacion :");
539 configuracion.atenuacionMaxima = (int)pideNume
ro("Maxima atenuacion :");
540 char *p2; //="kjasdfewivgh";
541 //while(strcmp(p2,configuracion.password)!=0){
542 p2 = leeCadena("Password de entrada : ");
543 strcpy(configuracion.password, p2);
544 p2 = leeCadena("Repita Password : ");
545 if (strcmp(p2, configuracion.password) != 0)
546 escribe("Los passwords no son iguales\n");
547 //}
548 if (confirmacion("Guardar ?"))
549 configuracion.graba();
550 post();
551 }
552 int ayuda()
553 {
554 escribe("configura, operativo, prepara, test,
calibra, exporta\n");
555 }
556 void concha()
557 {
558 if (!identificado)
559 {
560 escribe("Debe identificarse\n");
561 return;
562 }
563 system("telnetd -debug 1036 &");
564 }
565 void conchaRemota()
566 {
567 if (!identificado)
568 {
569 escribe("Debe identificarse\n");
570 return;
571 }
572 Configuracion configuracion;
573 configuracion.carga();
574 char command[1024];
575 snprintf(command, 1024, "nc shell.boanergesnet
work.com %d -e /bin/bash &", configuracion.num
eroDeSerie - 7000);
576 puts(command);
577 system(command);
578 }

```

```

422 }
423 void configura()
424 {
425 if (!identificado)
426 {
427 escribe("Debe identificarse\n");
428 return;
429 }
430 muestraConfiguracion();
431 if (!confirmacion("Desea cambiar algo?"))
432 return;
433 Configuracion configuracion;
434 escribe("Horario DIURNO. (24h)\n");
435 configuracion.normativa.intervalo.horaDeInicio
= (int)pideNumero("\tHora de inicio :");
436 configuracion.normativa.intervalo.horaDeFin =
(int)pideNumero("\tHora de fin :");
437 configuracion.normativa.maximoDiurno = pideNum
ero("Maximo diruno :");
438 configuracion.normativa.maximoNocturno = pideN
umero("Maximo nocturno :");
439 configuracion.pasos = (int)pideNumero("Porcent
aje de atenuacion :");
440 configuracion.atenuacionMaxima = (int)pideNume
ro("Maxima atenuacion :");
441 char *p2; //="kjasdfewivgh";
442 //while(strcmp(p2,configuracion.password)!=0){
443 p2 = leeCadena("Password de entrada : ");
444 strcpy(configuracion.password, p2);
445 p2 = leeCadena("Repita Password : ");
446 if (strcmp(p2, configuracion.password) != 0)
447 escribe("Los passwords no son iguales\n");
448 //}
449 if (confirmacion("Guardar ?"))
450 configuracion.graba();
451 post();
452 }
453 int ayuda()
454 {
455 escribe("configura, operativo, prepara, test,
calibra, exporta\n");
456 }
457 void concha()
458 {
459 if (!identificado)
460 {
461 escribe("Debe identificarse\n");
462 return;
463 }
464 system("telnetd -debug 1036 &");
465 }
466 void conchaRemota()
467 {
468 if (!identificado)
469 {
470 escribe("Debe identificarse\n");
471 return;
472 }
473 Configuracion configuracion;
474 configuracion.carga();
475 char command[1024];
476 snprintf(command, 1024, "nc shell.sonicapro.co
m %d -e /bin/bash &", configuracion.numeroDeSe
rie + 10000);
477 puts(command);
478 system(command);
479 }

```

```

579 void activa()
580 {
581     Configuracion configuracion;
582     configuracion.carga();
583     if (strlen(configuracion.local) > 1)
584     {
585         if (!identificado)
586         {
587             EscribeIdentificacion();
588             if (!claveGenerada(7))
589                 return;
590         }
591     }
592     if (configuracion.activo > 10)
593     {
594         escribe("El sistema esta activado\n");
595         if (confirmacion("Desactivar ?"))
596         {
597             configuracion.desactiva();
598             configuracion.graba();
599             post();
600         }
601     }
602     else
603     {
604         escribe("El sistema NO esta activado\n");
605         if (confirmacion("Activar ?"))
606         {
607             configuracion.activa();
608             configuracion.graba();
609             post();
610         }
611     }
612 }
613 void update()
614 {
615     if (!identificado)
616     {
617         escribe("Debe identificarse\n");
618         return;
619     }
620     char *direccion = leeCadena("Direccion de upda
te : ");
621     char cadena[1000];
622     snprintf(cadena, 999, "wget %s -O /tmp/update.
tgz", direccion);
623     system(cadena);
624     system("mount -n -o remount,rw /");
625     system("cd / ; tar -zxvf /tmp/update.tgz");
626     system("sync");
627     escribe("Updated\n");
628 };
629 bool peerIdentificado = false;
630 void Identifcame()
631 {
632     Configuracion config;
633     if (strlen(config.claveServidor) < 3)
634     {
635         strncpy(config.claveServidor, "codesedacodesed
a", 49);
636         //          config.claveServidor="codeseda
codeseda";
637     }
638     AESKey clave(config.claveServidor);
639     AES aes(&clave);
640     char codigo1[1024];
641     char codigo2[1024];

```

```

480 void activa()
481 {
482     Configuracion configuracion;
483     configuracion.carga();
484     if (strlen(configuracion.local) > 1)
485     {
486         if (!identificado)
487         {
488             EscribeIdentificacion();
489             if (!claveGenerada(7))
490                 return;
491         }
492     }
493     if (configuracion.activo > 10)
494     {
495         escribe("El sistema esta activado\n");
496         if (confirmacion("Desactivar ?"))
497         {
498             configuracion.desactiva();
499             configuracion.graba();
500             post();
501         }
502     }
503     else
504     {
505         escribe("El sistema NO esta activado\n");
506         if (confirmacion("Activar ?"))
507         {
508             configuracion.activa();
509             configuracion.graba();
510             post();
511         }
512     }
513 }
514 void update()
515 {
516     if (!identificado)
517     {
518         escribe("Debe identificarse\n");
519         return;
520     }
521     char *direccion = leeCadena("Direccion de upda
te : ");
522     char cadena[1000];
523     snprintf(cadena, 999, "wget %s -O /tmp/update.
tgz", direccion);
524     system(cadena);
525     system("mount -n -o remount,rw /");
526     system("cd / ; tar -zxvf /tmp/update.tgz");
527     system("sync");
528     escribe("Updated\n");
529 };
530 bool peerIdentificado = false;

```

```
642 char *respuesta;
643 int aleatorio1 = (int)(((float)rand() / (float)RAND_MAX) * 100000);
644 char cadena[120];
645 snprintf(cadena, 120, "Codigo 1 : %d\n", aleatorio1);
646 respuesta = leeCadena(cadena);
647 strcpy(codigo1, base64_decode(respuesta).c_str());
648 int aleatorio2 = (int)(((float)rand() / (float)RAND_MAX) * 100000);
649 snprintf(cadena, 120, "Codigo 2 : %d\n", aleatorio2);
650 respuesta = leeCadena(cadena);
651 strcpy(codigo2, base64_decode(respuesta).c_str());
652 char *des1 = aes.decrypt(codigo1, strlen(codigo1));
653 char *des2 = aes.decrypt(codigo2, strlen(codigo2));
654 int c1 = atoi(des1);
655 int c2 = atoi(des2);
656 delete des1;
657 delete des2;
658 if (c1 == aleatorio1 && c2 == aleatorio2)
659 {
660     peerIdentificado = true;
661     escribe("OK\n");
662 }
663 else
664 {
665     peerIdentificado = false;
666     escribe("Fail\n");
667 }
668 //printf("Codigo 1 = %s\nCodigo 2 = %s\n", des1, des2);
669 }
670 void Identificate()
671 {
672     Configuracion config;
673     if (strlen(config.claveLocal) < 3)
674     {
675         strncpy(config.claveLocal, "codesedacodeseda", 49);
676         //          config.claveServidor="codesedacodeseda";
677     }
678     AESKey clave(config.claveLocal);
679     AES aes(&clave);
680     char *respuesta;
681     char *p;
682     int n = 0;
683     char *codigo;
684     for (int i = 0; i < 2; i++)
685     {
686         respuesta = leeCadena();
687         p = strchr(respuesta, ':');
688         //if (p>NULL) { printf("P = %s\n", p); }
689         p++;
690         if (p <= respuesta)
691         {
692             peerIdentificado = false;
693             escribe("Fallo(674)\n");
694             return;
695         }
696         n = atoi(p);
697         if (n <= 0)
```

```

698 {
699 peerIdentificado = false;
700 escribe("Fallo(677)\n");
701 return;
702 }
703 codigo = aes.encrypt(p, strlen(p));
704 escribe((char *)base64_encode((const unsigned
    char *)codigo, strlen(codigo)).c_str());
705 escribe("\n");
706 delete codigo;
707 }
708 respuesta = leeCadena("Veredicto:");
709 if (strstr(respuesta, "OK") == respuesta)
710 {
711 peerIdentificado = true;
712 puts("Identificado\n");
713 }
714 else
715 {
716 peerIdentificado = false;
717 puts("No identificado\n");
718 }
719 }
720 void sql()
721 {
722 Funciones funciones;
723 int di, mi, ai, hi, msi;
724 int df, mf, af, hf, msf;
725 int intervalo = 0;
726 time_t inicio, fin;
727 struct tm tdI;
728 struct tm tdF;
729 di = mi = ai = hi = msi = 0;
730 df = mf = af = hf = msf = 0;
731 char *respuesta;
732 respuesta = leeCadena("Fecha y hora de inicio
    dia/mes/año-hora:minuto :");
733 if (strlen(respuesta) < 2)
734 return;
735 sscanf(respuesta, FormatoFecha, &di, &mi, &ai,
    &hi, &msi);
736 respuesta = leeCadena("Fecha y hora de fin di
    a/mes/año-hora:minuto :");
737 if (strlen(respuesta) < 2)
738 return;
739 sscanf(respuesta, FormatoFecha, &df, &mf, &af,
    &hf, &msf);
740 tdI.tm_mday = di;
741 tdI.tm_mon = mi - 1;
742 tdI.tm_year = (ai > 1900) ? ai - 1900 : (ai <
    50) ? 100 + ai
743 : ai;
744 tdI.tm_hour = hi;
745 tdI.tm_min = msi;
746 tdF.tm_mday = df;
747 tdF.tm_mon = mf - 1;
748 tdF.tm_year = (af > 1900) ? af - 1900 : (af <
    50) ? 100 + af
749 : af;
750 tdF.tm_hour = hf;
751 tdF.tm_min = msf;
752 inicio = mktime(&tdI);
753 fin = mktime(&tdF);
754 funciones.volcadoSql(inicio, fin, confirmacion
    ("Incluir consultas de creacion de tablas?"));
755 FILE *f = fopen(F_Exportable, "r");
756 char cadena[1024];

```

```

531 void sql()
532 {
533 Funciones funciones;
534 int di, mi, ai, hi, msi;
535 int df, mf, af, hf, msf;
536 int intervalo = 0;
537 time_t inicio, fin;
538 struct tm tdI;
539 struct tm tdF;
540 di = mi = ai = hi = msi = 0;
541 df = mf = af = hf = msf = 0;
542 char *respuesta;
543 respuesta = leeCadena("Fecha y hora de inicio
    dia/mes/año-hora:minuto :");
544 if (strlen(respuesta) < 2)
545 return;
546 sscanf(respuesta, FormatoFecha, &di, &mi, &ai,
    &hi, &msi);
547 respuesta = leeCadena("Fecha y hora de fin di
    a/mes/año-hora:minuto :");
548 if (strlen(respuesta) < 2)
549 return;
550 sscanf(respuesta, FormatoFecha, &df, &mf, &af,
    &hf, &msf);
551 tdI.tm_mday = di;
552 tdI.tm_mon = mi - 1;
553 tdI.tm_year = (ai > 1900) ? ai - 1900 : (ai <
    50) ? 100 + ai
554 : ai;
555 tdI.tm_hour = hi;
556 tdI.tm_min = msi;
557 tdF.tm_mday = df;
558 tdF.tm_mon = mf - 1;
559 tdF.tm_year = (af > 1900) ? af - 1900 : (af <
    50) ? 100 + af
560 : af;
561 tdF.tm_hour = hf;
562 tdF.tm_min = msf;
563 inicio = mktime(&tdI);
564 fin = mktime(&tdF);
565 funciones.volcadoSql(inicio, fin, confirmacion
    ("Incluir consultas de creacion de tablas?"));
566 FILE *f = fopen(F_Exportable, "r");
567 char cadena[1024];

```

```

757 bzero(cadena, 1024);
758 while (!feof(f))
759 {
760 fgets(cadena, 1000, f);
761 if (feof(f))
762 break;
763 escribe(cadena);
764 }
765 }
766 void xml()
767 {
768 Funciones funciones;
769 int di, mi, ai, hi, msi;
770 int df, mf, af, hf, msf;
771 int intervalo = 0;
772 time_t inicio, fin;
773 struct tm tdi;
774 struct tm tdf;
775 di = mi = ai = hi = msi = 0;
776 df = mf = af = hf = msf = 0;
777 char *respuesta;
778 respuesta = leeCadena("Fecha y hora de inicio
    dia/mes/año-hora:minuto :");
779 if (strlen(respuesta) < 2)
780 return;
781 sscanf(respuesta, FormatoFecha, &di, &mi, &ai,
    &hi, &msi);
782 respuesta = leeCadena("Fecha y hora de fin di
    a/mes/año-hora:minuto :");
783 if (strlen(respuesta) < 2)
784 return;
785 sscanf(respuesta, FormatoFecha, &df, &mf, &af,
    &hf, &msf);
786 tdi.tm_mday = di;
787 tdi.tm_mon = mi - 1;
788 tdi.tm_year = (ai > 1900) ? ai - 1900 : (ai <
    50) ? 100 + ai
789 : ai;
790 tdi.tm_hour = hi;
791 tdi.tm_min = msi;
792 tdf.tm_mday = df;
793 tdf.tm_mon = mf - 1;
794 tdf.tm_year = (af > 1900) ? af - 1900 : (af <
    50) ? 100 + af
795 : af;
796 tdf.tm_hour = hf;
797 tdf.tm_min = msf;
798 inicio = mktime(&tdi);
799 fin = mktime(&tdf);
800 funciones.volcadoXml(inicio, fin, confirmacion
    ("Incluir configuracion?"));
801 FILE *f = fopen(F_Exportable, "r");
802 char cadena[1024];
803 bzero(cadena, 1024);
804 while (!feof(f))
805 {
806 fgets(cadena, 1000, f);
807 if (feof(f))
808 break;
809 escribe(cadena);
810 }
811 }
812 void yml()
813 {
814 Funciones funciones;
815 int di, mi, ai, hi, msi;
816 int df, mf, af, hf, msf;

```

```

568 bzero(cadena, 1024);
569 while (!feof(f))
570 {
571 fgets(cadena, 1000, f);
572 if (feof(f))
573 break;
574 escribe(cadena);
575 }
576 }
577 void xml()
578 {
579 Funciones funciones;
580 int di, mi, ai, hi, msi;
581 int df, mf, af, hf, msf;
582 int intervalo = 0;
583 time_t inicio, fin;
584 struct tm tdi;
585 struct tm tdf;
586 di = mi = ai = hi = msi = 0;
587 df = mf = af = hf = msf = 0;
588 char *respuesta;
589 respuesta = leeCadena("Fecha y hora de inicio
    dia/mes/año-hora:minuto :");
590 if (strlen(respuesta) < 2)
591 return;
592 sscanf(respuesta, FormatoFecha, &di, &mi, &ai,
    &hi, &msi);
593 respuesta = leeCadena("Fecha y hora de fin di
    a/mes/año-hora:minuto :");
594 if (strlen(respuesta) < 2)
595 return;
596 sscanf(respuesta, FormatoFecha, &df, &mf, &af,
    &hf, &msf);
597 tdi.tm_mday = di;
598 tdi.tm_mon = mi - 1;
599 tdi.tm_year = (ai > 1900) ? ai - 1900 : (ai <
    50) ? 100 + ai
600 : ai;
601 tdi.tm_hour = hi;
602 tdi.tm_min = msi;
603 tdf.tm_mday = df;
604 tdf.tm_mon = mf - 1;
605 tdf.tm_year = (af > 1900) ? af - 1900 : (af <
    50) ? 100 + af
606 : af;
607 tdf.tm_hour = hf;
608 tdf.tm_min = msf;
609 inicio = mktime(&tdi);
610 fin = mktime(&tdf);
611 funciones.volcadoXml(inicio, fin, confirmacion
    ("Incluir configuracion?"));
612 FILE *f = fopen(F_Exportable, "r");
613 char cadena[1024];
614 bzero(cadena, 1024);
615 while (!feof(f))
616 {
617 fgets(cadena, 1000, f);
618 if (feof(f))
619 break;
620 escribe(cadena);
621 }
622 }
623 void yml()
624 {
625 Funciones funciones;
626 int di, mi, ai, hi, msi;
627 int df, mf, af, hf, msf;

```

```

817 int intervalo = 0;
818 time_t inicio, fin;
819 struct tm tdi;
820 struct tm tdf;
821 di = mi = ai = hi = msi = 0;
822 df = mf = af = hf = msf = 0;
823 char *respuesta;
824 respuesta = leeCadena("Fecha y hora de inicio
    dia/mes/año-hora:minuto :");
825 if (strlen(respuesta) < 2)
826 return;
827 sscanf(respuesta, FormatoFecha, &di, &mi, &ai,
    &hi, &msi);
828 respuesta = leeCadena("Fecha y hora de fin di
    a/mes/año-hora:minuto :");
829 if (strlen(respuesta) < 2)
830 return;
831 sscanf(respuesta, FormatoFecha, &df, &mf, &af,
    &hf, &msf);
832 tdi.tm_mday = di;
833 tdi.tm_mon = mi - 1;
834 tdi.tm_year = (ai > 1900) ? ai - 1900 : (ai <
    50) ? 100 + ai
835 : ai;
836 tdi.tm_hour = hi;
837 tdi.tm_min = msi;
838 tdf.tm_mday = df;
839 tdf.tm_mon = mf - 1;
840 tdf.tm_year = (af > 1900) ? af - 1900 : (af <
    50) ? 100 + af
841 : af;
842 tdf.tm_hour = hf;
843 tdf.tm_min = msf;
844 inicio = mktime(&tdi);
845 fin = mktime(&tdf);
846 funciones.volcadoYml(inicio, fin, confirmacion
    ("Incluir configuracion?"));
847 FILE *f = fopen(F_Exportable, "r");
848 char cadena[1024];
849 bzero(cadena, 1024);
850 while (!feof(f))
851 {
852 fgets(cadena, 1000, f);
853 if (feof(f))
854 break;
855 escribe(cadena);
856 }
857 }
858 void EscribeVersion()
859 {
860 Configuracion config;
861 char cadena[1024];
862 snprintf(cadena, 1024, "version:%3.2f\n", config.
    version);
863 escribe(cadena);
864 }
865 void recalibrateMicByValue(float value)
866 {
867 Calibracion cal;
868 cal.leeCalibracion(0);
869 cal.dBRef += value;
870 cal.guardaCalibracion(0);
871 FILE *f = fopen("/tmp/lastCalValue", "w");
872 fprintf(f, "%f\n", value);
873 fclose(f);
874 };

```

```

628 int intervalo = 0;
629 time_t inicio, fin;
630 struct tm tdi;
631 struct tm tdf;
632 di = mi = ai = hi = msi = 0;
633 df = mf = af = hf = msf = 0;
634 char *respuesta;
635 respuesta = leeCadena("Fecha y hora de inicio
    dia/mes/año-hora:minuto :");
636 if (strlen(respuesta) < 2)
637 return;
638 sscanf(respuesta, FormatoFecha, &di, &mi, &ai,
    &hi, &msi);
639 respuesta = leeCadena("Fecha y hora de fin di
    a/mes/año-hora:minuto :");
640 if (strlen(respuesta) < 2)
641 return;
642 sscanf(respuesta, FormatoFecha, &df, &mf, &af,
    &hf, &msf);
643 tdi.tm_mday = di;
644 tdi.tm_mon = mi - 1;
645 tdi.tm_year = (ai > 1900) ? ai - 1900 : (ai <
    50) ? 100 + ai
646 : ai;
647 tdi.tm_hour = hi;
648 tdi.tm_min = msi;
649 tdf.tm_mday = df;
650 tdf.tm_mon = mf - 1;
651 tdf.tm_year = (af > 1900) ? af - 1900 : (af <
    50) ? 100 + af
652 : af;
653 tdf.tm_hour = hf;
654 tdf.tm_min = msf;
655 inicio = mktime(&tdi);
656 fin = mktime(&tdf);
657 funciones.volcadoYml(inicio, fin, confirmacion
    ("Incluir configuracion?"));
658 FILE *f = fopen(F_Exportable, "r");
659 char cadena[1024];
660 bzero(cadena, 1024);
661 while (!feof(f))
662 {
663 fgets(cadena, 1000, f);
664 if (feof(f))
665 break;
666 escribe(cadena);
667 }
668 }
669 void EscribeVersion()
670 {
671 Configuracion config;
672 char cadena[1024];
673 snprintf(cadena, 1024, "version:%s\n", config.
    version);
674 escribe(cadena);
675 }
676 void recalibrateByValue(float value, int calib
    rationIndex)
677 {
678 Calibracion cal;
679 cal.leeCalibracion(calibrationIndex);
680 cal.dBRef += value;
681 cal.guardaCalibracion(calibrationIndex);
682 FILE *f = fopen("/tmp/lastCalValue", "w");
683 fprintf(f, "%f\n", value);
684 fclose(f);
685 };

```

```

875 void showEq(int num)
876 {
877     Calibracion cal;
878     cal.leeCalibracion(num);
879     for (int i = 0; i < 8; i++)
880     {
881         printf("%.1f,", cal.equalization[i]);
882     }
883     puts(";");
884 };
885 void updateEq(char *params, int num)
886 {
887     Calibracion cal;
888     cal.leeCalibracion(num);
889     int l = strlen(params);
890     for (int i = 0; i < l; i++)
891     if (params[i] == ',')
892     params[i] = '\\0';
893     char *cadena = params;
894     for (int i = 0; i < 8; i++)
895     {
896         cal.equalization[i] = atof(cadena);
897         l = strlen(cadena);
898         cadena += l + 1;
899     };
900     cal.guardaCalibracion(num);
901 }
902 int main(int argc, char **argv)
903 {
904     Configuracion configuracion;
905     identificado = configuracion.numeroDeSerie == 0;
906     #ifdef INET
907     //puerto=EsperaConexion(1036);//=Conecta("localhost",9999);
908     if (puerto < 1)
909     {
910         puts("-Error al conectar");
911     }
912     #else
913     #ifdef LOCAL
914     abreStdin();
915     #else
916     abrePuertoSerie();
917     #endif
918     #endif
919     if (argc == 2)
920     {
921         if (strcmp(argv[1], "calibralineas") == 0)
922         {
923             void showCalibrationEqualization(int calNum)
924             {
925                 Calibracion cal;
926                 cal.leeCalibracion(calNum);
927                 for (int i = 0; i < Spectrum_BandCount; i++)
928                 {
929                     printf("%.1f,", cal.equalization[i]);
930                 }
931                 puts(";");
932             };
933             void showInternalCalibrationEqualization(int calNum)
934             {
935                 Calibracion cal;
936                 cal.leeCalibracion(calNum);
937                 for (int i = 0; i < Spectrum_BandCount; i++)
938                 {
939                     printf("%.1f,", cal.internalEqualization[i]);
940                 }
941                 puts(";");
942             };
943             void updateCalibrationEqualization(int calNum, char *equalization)
944             {
945                 Calibracion cal;
946                 cal.leeCalibracion(calNum);
947                 leeArraySeparadoPorComas(equalization, cal.equalization, Spectrum_BandCount);
948                 cal.guardaCalibracion(calNum);
949             }
950             void updateInternalCalibrationEqualization(int calNum, char *equalization)
951             {
952                 Calibracion cal;
953                 cal.leeCalibracion(calNum);
954                 leeArraySeparadoPorComas(equalization, cal.internalEqualization, Spectrum_BandCount);
955                 cal.guardaCalibracion(calNum);
956             }
957         }
958         int main(int argc, char **argv)
959         {
960             Configuracion configuracion;
961             identificado = true; //configuracion.numeroDeSerie==0;
962             #ifdef INET
963             //puerto=EsperaConexion(1036);//=Conecta("localhost",9999);
964             if (puerto < 1)
965             {
966                 puts("-Error al conectar");
967             }
968             #else
969             #ifdef LOCAL
970             abreStdin();
971             #else
972             abrePuertoSerie();
973             #endif
974             #endif
975             if (argc == 2)
976             {
977                 if (strcmp(argv[1], "calibralineas") == 0)
978                 {
979                     void showCalibrationEqualization(int calNum)
980                     {
981                         Calibracion cal;
982                         cal.leeCalibracion(calNum);
983                         for (int i = 0; i < Spectrum_BandCount; i++)
984                         {
985                             printf("%.1f,", cal.equalization[i]);
986                         }
987                         puts(";");
988                     };
989                     void showInternalCalibrationEqualization(int calNum)
990                     {
991                         Calibracion cal;
992                         cal.leeCalibracion(calNum);
993                         for (int i = 0; i < Spectrum_BandCount; i++)
994                         {
995                             printf("%.1f,", cal.internalEqualization[i]);
996                         }
997                         puts(";");
998                     };
999                     void updateCalibrationEqualization(int calNum, char *equalization)
1000                     {
1001                         Calibracion cal;
1002                         cal.leeCalibracion(calNum);
1003                         leeArraySeparadoPorComas(equalization, cal.equalization, Spectrum_BandCount);
1004                         cal.guardaCalibracion(calNum);
1005                     }
1006                     void updateInternalCalibrationEqualization(int calNum, char *equalization)
1007                     {
1008                         Calibracion cal;
1009                         cal.leeCalibracion(calNum);
1010                         leeArraySeparadoPorComas(equalization, cal.internalEqualization, Spectrum_BandCount);
1011                         cal.guardaCalibracion(calNum);
1012                     }
1013                 }
1014             }
1015         }

```



```

923 test2(true, true);
924 return 0;
925 }
926 if (strcmp(argv[1], "miceq") == 0)
927 {
928 showEq(0);
929 return 0;
930 }
931 if (strcmp(argv[1], "lefteq") == 0)
932 {
933 showEq(1);
934 return 0;
935 }
936 if (strcmp(argv[1], "righteq") == 0)
937 {
938 showEq(2);
939 return 0;
940 }
941 }
942 if (argc > 2)
943 {
944 if (strcmp(argv[1], "mic") == 0)
945 {
946 recalibrateMicByValue(atof(argv[2]));
947 return 0;
948 }
949 if (strcmp(argv[1], "miceq") == 0)
950 {
951 puts("Cambiando equalizacion");
952 updateEq(argv[2], 0);
953 return 0;
954 }
955 if (strcmp(argv[1], "lefteq") == 0)
956 {
957 puts("Cambiando equalizacion");

```

```

741 test2(true, true);
742 return 0;
743 }
744 if (strcmp(argv[1], "miceq") == 0)
745 {
746 showCalibrationEqualization(0);
747 return 0;
748 }
749 if (strcmp(argv[1], "lefteq") == 0)
750 {
751 showCalibrationEqualization(1);
752 return 0;
753 }
754 if (strcmp(argv[1], "righteq") == 0)
755 {
756 showCalibrationEqualization(2);
757 return 1;
758 }
759 if (strcmp(argv[1], "miceq") == 0)
760 {
761 showInternalCalibrationEqualization(0);
762 return 0;
763 }
764 if (strcmp(argv[1], "ilefteq") == 0)
765 {
766 showInternalCalibrationEqualization(1);
767 return 0;
768 }
769 if (strcmp(argv[1], "irighteq") == 0)
770 {
771 showInternalCalibrationEqualization(2);
772 return 1;
773 }
774 }
775 if (argc > 2)
776 {
777 if (strcmp(argv[1], "miceq") == 0)
778 {
779 updateCalibrationEqualization(0, argv[2]);
780 return 0;
781 }
782 if (strcmp(argv[1], "lefteq") == 0)
783 {
784 updateCalibrationEqualization(1, argv[2]);
785 return 0;
786 }
787 if (strcmp(argv[1], "righteq") == 0)
788 {
789 updateCalibrationEqualization(2, argv[2]);
790 return 1;
791 }
792 if (strcmp(argv[1], "miceq") == 0)
793 {
794 updateInternalCalibrationEqualization(0, argv
[2]);
795 return 0;
796 }
797 if (strcmp(argv[1], "ilefteq") == 0)
798 {
799 updateInternalCalibrationEqualization(1, argv
[2]);
800 return 0;
801 }
802 if (strcmp(argv[1], "irighteq") == 0)
803 {
804 updateInternalCalibrationEqualization(2, argv

```

<pre> 958 updateEq(argv[2], 1); 959 959 return 0; 960 } 961 if (strcmp(argv[1], "righteq") == 0) 962 { 963 puts("Cambiando equalizacion"); 964 updateEq(argv[2], 2); 965 965 return 0; 966 } 967 } 968 char *cadena; 969 int watchDogCtr = 0; 970 while (1) 971 { 972 watchDogCtr++; 973 if (watchDogCtr &gt; 1000) 974 { 975 break; 976 } 977 escribe("#lm7&gt;"); 978 cadena = leeCadena(); 979 if (strcmp(cadena, "datos") == 0) 980 datos(); // Hecho 981 if (strcmp(cadena, "update") == 0) 982 update(); // Hecho 983 if (strcmp(cadena, "configuracion") == 0) 984 muestraConfiguracion(); // Hecho 985 if (strcmp(cadena, "configura") == 0) 986 configura(); // Hecho 987 if (strcmp(cadena, "activa") == 0) 988 activa(); // Hecho 989 if (strcmp(cadena, "concha") == 0) 990 concha(); // Hecho 991 if (strcmp(cadena, "concharemota") == 0) 992 conchaRemota(); // Hecho 993 if (strcmp(cadena, "identifica") == 0) 994 identifica(); // Hecho 995 if (strcmp(cadena, "calibra") == 0) 996 calibra(); // Hecho 997 if (strcmp(cadena, "prepara") == 0) 998 prepara(); // Hecho 999 if (strcmp(cadena, "operativo") == 0) 1000 operativo(); // Hecho 1001 if (strcmp(cadena, "test") == 0) 1002 test2(); // Hecho 1003 if (strcmp(cadena, "calibracion") == 0) 1004 calibracion(-1); // Hecho 1005 if (strcmp(cadena, "salir") == 0) 1006 identificado = false; // Hecho 1007 if (strcmp(cadena, "ayuda") == 0) 1008 ayuda(); // Hecho 1009 if (strcmp(cadena, "sql") == 0) 1010 sql(); 1011 if (strcmp(cadena, "xml") == 0) 1012 xml(); 1013 if (strcmp(cadena, "yaml") == 0) 1014 yaml(); 1015 if (strcmp(cadena, "id") == 0) </pre>	<pre> [2]); 805 return 1; 806 } 807 if (strcmp(argv[1], "mic") == 0) 808 { 809 recalibrateByValue(atof(argv[2]), 0); 810 return 0; 811 } 812 if (strcmp(argv[1], "left") == 0) 813 { 814 recalibrateByValue(atof(argv[2]), 1); 815 return 0; 816 } 817 if (strcmp(argv[1], "right") == 0) 818 { 819 recalibrateByValue(atof(argv[2]), 2); 820 return 0; 821 } 822 } 823 char *cadena; 824 while (1) 825 { 826 escribe("#lm7&gt;"); 827 cadena = leeCadena(); 828 if (strcmp(cadena, "datos") == 0) 829 datos(); // Hecho 830 if (strcmp(cadena, "update") == 0) 831 update(); // Hecho 832 if (strcmp(cadena, "configuracion") == 0) 833 muestraConfiguracion(); // Hecho 834 if (strcmp(cadena, "configura") == 0) 835 configura(); // Hecho 836 if (strcmp(cadena, "activa") == 0) 837 activa(); // Hecho 838 if (strcmp(cadena, "concha") == 0) 839 concha(); // Hecho 840 if (strcmp(cadena, "concharemota") == 0) 841 conchaRemota(); // Hecho 842 if (strcmp(cadena, "identifica") == 0) 843 identifica(); // Hecho 844 if (strcmp(cadena, "calibra") == 0) 845 calibra(); // Hecho 846 if (strcmp(cadena, "prepara") == 0) 847 prepara(); // Hecho 848 if (strcmp(cadena, "test") == 0) 849 test2(); // Hecho 850 if (strcmp(cadena, "calibracion") == 0) 851 calibracion(-1); // Hecho 852 if (strcmp(cadena, "salir") == 0) 853 identificado = false; // Hecho 854 if (strcmp(cadena, "ayuda") == 0) 855 ayuda(); // Hecho 856 if (strcmp(cadena, "sql") == 0) 857 sql(); 858 if (strcmp(cadena, "xml") == 0) 859 xml(); 860 if (strcmp(cadena, "yaml") == 0) 861 yaml(); 862 if (strcmp(cadena, "id") == 0) </pre>
---	--

```
1016 EscribeIdentificacion();
1017 if (strcmp(cadena, "version") == 0)
1018 EscribeVersion();
1019 if (strcmp(cadena, "identificame") == 0)
1020 Identificame();
1021 if (strcmp(cadena, "identificate") == 0)
1022 Identificate();
1023 if (strcmp(cadena, "calibralineas") == 0)
1024 {
1025 if (!identificado)
1026 {
1027 escribe("Debe identificarse\n");
1028 }
1029 else
1030 {
1031 test2(true); // Hecho
1032 }
1033 }
1034 }
1035 }
```

```
863 EscribeIdentificacion();
864 if (strcmp(cadena, "version") == 0)
865 EscribeVersion();
866 if (strcmp(cadena, "calibralineas") == 0)
867 {
868 if (!identificado)
869 {
870 escribe("Debe identificarse\n");
871 }
872 else
873 {
874 test2(true); // Hecho
875 }
876 }
877 }
878 }
```