sep 06, 16 18:42 **time.h** Page 1/1

```h
#ifndef TIME_H
#define TIME_H



//Funciones
int wait(int (*lista)[6]);
void getHrMinSec(char* hora,int (*lista)[6]);
void printTime(int (*list)[6]);
int incrementMinute(int *minutes);

#endif
```

sep 06, 16 18:42 **time.c** Page 1/2

```c
#include <stdio.h> //printf() sscanf()
#include <time.h> //time()
//#include <stdlib.h> //atoi()
#include <unistd.h> //sleep()

//Este mÃ³dulo se encarga de obtener la hora y si los segundos son igual a 0
//entonces por STDIN

// {aÃ±o,mes,dia,hora,minutos,segundos}
//   0   1   2   3      4       5
void printTime(int (*list)[6]){
    fprintf(stderr,"%d.%02d.%02d-%02d:%02d:00 - ",(*list)[0],(*list)[1],
    (*list)[2],(*list)[3],(*list)[4]);
  }

int getHrMinSec(char* hora,int (*lista)[6]){
  int anio,mes,dia,hours,minutes,seconds;
  sscanf(hora,"%d.%d.%d-%d:%d:%d",&anio,&mes,&dia,&hours,&minutes,&seconds);
  (*lista)[0] = anio;
  (*lista)[1] = mes;
  (*lista)[2] = dia;
  (*lista)[3] = hours;
  (*lista)[4] = minutes;
  (*lista)[5] = seconds;
  return 0;
}


int incrementMinute(int (*lista)[6]){
  //printf("(*lista)[4]:%d\n",(*lista)[4]);
  if ( (*lista)[4] < 59 ){
    (*lista)[4] = (*lista)[4] + 1;
    return 1;
  }else{
    (*lista)[4] = 0;
    return 0;
  }
}

int incrementHour(int (*lista)[6]){
  if ((*lista)[3] < 23){
    //printf("Sumo uno a la hora\n");
    (*lista)[3] = (*lista)[3] + 1;
    return 1;
  }else{
    (*lista)[3] = 0;
    //printf("La hora es cero\n");
    return 0;
  }
}

int incrementDay(int (*lista)[6]){
  //printf("Incremento dia\n");
  if ((*lista)[2] < 31){
    (*lista)[2] = (*lista)[2] + 1;
    return 1;
  }else{
    (*lista)[2] = 1;
    return 0;
  }
}

int wait(int (*lista)[6]){
  int cambioHora = 1;
  //printf("wait");
  (*lista)[5] = 0;
```

sep 06, 16 18:42 **time.c** Page 2/2

```
67    if (incrementMinute(lista) == 0){
68      cambioHora = incrementHour(lista);
69    }
70    if (cambioHora == 0){
71      incrementDay(lista);
72    }
73    //printTime(lista);
74    return 0;
75  }
```

sep 06, 16 18:42 **server.h** Page 1/1

```
1   #ifndef SERVER_H
2   #define SERVER_H
3
4   #include <stdio.h>
5   #include <stdlib.h>
6   #include <sys/socket.h>
7   #include <string.h> //srtlen()
8
9
10  #include "aceptador.h"
11  #include "conectador.h"
12  #include "time.h"
13  #include "lista.h"
14
15  int server(int argc, char *argv[]);
16
17
18  #endif
```

```c
#include "server.h"

//calcula el mÃ¡ximo de las temperaturas
void getMax(float* maxActual,float* num){
  if ((*num) > (*maxActual)){
    *maxActual = *num;
  }
}

//actualiza el mÃ-nimo de las temperaturas
void getMin(float* minActual, float* num){
  if((*num)<(*minActual)){
    *minActual = *num;
  }
}

//actualiza la sumatoria de todas las temperaturas
void refreshSum(float* sumatoria,float* num){
  *sumatoria = *sumatoria + *num;
}

float getFloat(char* charFloat){
  return atof(charFloat);
}


void formatParamaters(float* max,float* min,lista_t* list,int *cantidad){
  *max = -18.0;
  *min = 60.0;
  while(¬lista_esta_vacia(list)){
    lista_borrar_primero(list);
  }
  //printf("formatParamaters");
  //printList(list);
  *cantidad = 0;
}

void printInfo(int (*list)[6],char* id,float* max,float* min,lista_t* lista,
int* cantidadPorDia, int resta){
  //int posicionMedio = getLargo(lista)/2;
  //printf("La posicion media es:%d\n",posicionMedio);
  float mediana = lista_posicion(lista,getLargo(lista)/2);
  printf("%d.%02d.%02d %s Max=%.1f Min=%.1f Mediana=%.1f Muestras=%d\n",
  (*list)[0],(*list)[1],(*list)[2]-resta,id,*max,*min,mediana,*cantidadPorDia);
  //printf("Imprimi la lista\n");
  //printList(lista);
  formatParamaters(max,min,lista,cantidadPorDia);
}



int detectChangeDay(char* dateTime, int (*lista)[6]){
  getHrMinSec(dateTime,lista);
  return (*lista)[3] ≡ 0 ∧ (*lista)[4]≡0;
}

int server_prepare_connect(char *prt, conectador_t **conec, aceptador_t **acep){
  int cant_client = 10;
  int port_aux = atoi(prt);
  aceptador_t * acept_aux = socket_acept_create();
  if (acept_aux ≡ NULL) {
    printf(" no obtuve el aceptador\n");
    return -1;}
  *acep = acept_aux;
  if(socket_acept_connect(acept_aux,port_aux ,cant_client,"127.0.0.1") ≡ -1){
    printf(" no pude prepararme para conectarme\n");
```



```c
    socket_acept_close(acept_aux);
    return -1;
  }
  if (socket_acept_accept(*acep, conec) ≡ -1){
    printf(" no pude conectar\n");
    socket_acept_close(acept_aux);
    return -1;}
  return 1;
}

//la funcion receive devuelve -1 cuando la conexion se cierra
int receive_time(conectador_t* canal,char* time_char,int largo){
  int resultado = socket_conectador_receive(canal,time_char,largo);
  //printf("resultado de receive_time:%d\n",resultado);
  return resultado;
}

int server_free_memory(conectador_t *conect,aceptador_t* acept,char* hour){
  socket_conectador_close(conect);
  socket_acept_close(acept);
  free(hour);
  return 0;
}


int server(int argc,char* argv[]){
  if (argc < 3) {
    printf("Falta argumentos\n");
    return -1;
  }
  conectador_t *canal;
  aceptador_t *aceptador;
  if (server_prepare_connect(argv[2],&canal,&aceptador) ≡ -1){
    printf("No pude conectarme con el cliente\n");
    return -1;
  }
  //recibo el id del termostato
  char id_termostato[7] = "";
  socket_conectador_receive(canal,id_termostato,7);
  fprintf(stderr,"Recibiendo termostato. ID=%s\n", id_termostato);

  //variables para el tiempo
  int long_format_time = 20;
  char* time_char = malloc(sizeof(char)*long_format_time);


  //variables para la temperatura

  int long_format_temp = 6;
  char* temp_char = malloc(sizeof(char)*long_format_temp);
  int long_format_ident = 1;
  char* identificador = malloc(sizeof(char)*long_format_ident);

  //variables para la informaciÃ³n a mostrar
  float max = -18.0;
  float min = 60.0;
  lista_t* lista = lista_crear();
  int cantidadPorMinuto = 0;
  int cantidadPorDia = 0;
  int arrayTime[6]; //aca voy guardado el date
  float number;
  //recibo el date junto con las mediciones
  while (receive_time(canal,time_char,long_format_time) ≠ -1){
    fprintf(stderr, "%s - ",time_char);
    //printf("%s - ", time_char);
    if (detectChangeDay(time_char,&arrayTime)){
```

sep 06, 16 18:42 **server.c** Page 3/3

```
133         printInfo(&arrayTime,id_termostato,&max,&min,lista,&cantidadPorDia,1);
134       }
135       strncpy(identificador," ",long_format_ident);
136       while (strncmp(identificador," ",long_format_ident) ≡ 0) {
137         socket_conectador_receive(canal,temp_char,long_format_temp);
138         number = getFloat(temp_char);
139         //printf("El numero flotante es:%f-", number);
140         getMin(&min,&number);
141         getMax(&max,&number);
142         lista_insertar(lista,number);
143         cantidadPorMinuto++;
144         cantidadPorDia++;
145         socket_conectador_receive(canal,identificador,long_format_ident);
146       }
147       //printf("");
148       //printf("%s",identificador);
149       fprintf(stderr,"Datos recibidos: %d\n",cantidadPorMinuto);
150       //printf(,cantidadPorMinuto);
151       cantidadPorMinuto = 0;
152     }
153     //refreshSum(&sumatoria,&number);
154     printInfo(&arrayTime,id_termostato,&max,&min,lista,&cantidadPorDia,0);
155     fprintf(stderr,"Termostato desconectado. ID=%s\n", id_termostato);
156     free(identificador);
157     free(temp_char);
158     lista_destruir(lista);
159
160     server_free_memory(canal,aceptador,time_char);
161     return 1;
162   }
```

sep 06, 16 18:42 **principal.c** Page 1/1

```
1   #include <stdio.h> //printf()
2   #include <string.h> //strcmp()
3   #include <stdlib.h> //malloc()
4
5   #include "time.h"
6   #include "client.h"
7   #include "server.h"
8
9   //supongo que el segundo argumento es el nombre del archivo
10  int main(int argc, char *argv[]){
11    //int n = 0;
12    //unsigned short int buffer;
13    //char hexa[5];
14    //char algo[3] = "AB";
15    //printf("La cantidad de bytes que ocupa algo es:%d\n",(int)sizeof(algo));
16    if (strcmp(argv[1],"client") ≡ 0){
17      //printf("Hola soy un cliente\n");
18      client(argc,argv);
19    }
20    if (strcmp(argv[1],"server") ≡ 0){
21      //printf("Hola soy un servidor\n");
22      server(argc,argv);
23    }
24    //file_t* file = file_open("values.dat","rb");
25    // while (file_read(file,&buffer) != 0){
26    //   n++;
27    //   buffer = htons(buffer);
28    //   printf("%04x ",buffer);
29    //   //sprintf(hexa,"%04x",buffer);
30    //   snprintf(hexa,sizeof(hexa),"%04x",buffer);
31    //   printf("El valor de hexa:%s\n",hexa);
32    // }
33    // printf("La cantidad de mediciones es:%d\n",n);
34    // file_close(file);
35    return 0;
36  }
```

sep 06, 16 18:42 **nodo.h** Page 1/1

```c
#ifndef NODO_H
#define NODO_H

#include <stdlib.h>
#include <stdbool.h>

typedef struct nodo nodo_t;

// Crea un nodo.
nodo_t* crear_nodo(float dato,nodo_t *referencia);
nodo_t* obtener_referencia(nodo_t *nodo);
float obtener_dato(nodo_t *nodo);
void definir_referencia(nodo_t *nodo, nodo_t * nodo_referencia);
void set_dato_nodo(nodo_t* nodo,float dato);

// Destruye el nodo
void destruir_nodo(nodo_t *nodo);


#endif
```

sep 06, 16 18:42 **nodo.c** Page 1/1

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include "lista.h"
#include <assert.h>
#include "nodo.h"

struct nodo{
  float dato;
  nodo_t *referencia;
  };

// Crea un nodo.
nodo_t* crear_nodo(float dato,nodo_t *referencia){
  nodo_t *nuevo_nodo = malloc(sizeof(nodo_t));
  if (nuevo_nodo ≡ NULL){
    return NULL;
    }
  nuevo_nodo→dato = dato;
  nuevo_nodo→referencia = referencia;
  return nuevo_nodo;
}

void set_dato_nodo(nodo_t* nodo,float dato_aux){
  nodo→dato = dato_aux;
  //printf("Cambiando el dato, ahora es:%f\n",nodo->dato);
}

void definir_referencia(nodo_t *nodo, nodo_t * nodo_referencia){
  nodo→referencia = nodo_referencia;
}

nodo_t* obtener_referencia(nodo_t *nodo){
  return nodo→referencia;
}

float obtener_dato(nodo_t *nodo){
  return nodo→dato;
}



// Destruye el nodo
void destruir_nodo(nodo_t *nodo){
  free(nodo);
}
```

sep 06, 16 18:42 **lista.h** Page 1/1

```c
#ifndef LISTA_ENLAZADA_H
#define LISTA_ENLAZADA_H

#include <stdbool.h>
#include <stddef.h>
#include "nodo.h"



typedef struct lista lista_t;


//estas son las funciones que voy a utilizar

lista_t *lista_crear();
bool lista_insertar(lista_t *lista, float dato);
float lista_posicion(lista_t *lista,int posicion);
int getLargo(lista_t* lista);
void printList(lista_t* lista);
//estas son necesarias de forma interna
bool lista_esta_vacia(const lista_t *lista);
void lista_destruir(lista_t *lista);
void lista_borrar_primero(lista_t *lista);




#endif
```

sep 06, 16 18:42 **lista.c** Page 1/2

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include "lista.h"
#include <assert.h>
//#include "nodo.h"

struct lista{
  size_t largo;
  nodo_t* primero;
  nodo_t* ultimo;
  };

//-----------------------------------------------------------------------------

lista_t *lista_crear(){
  lista_t *lista_nueva = malloc (sizeof(lista_t));
  if (lista_nueva ≡ NULL){
    return NULL;
    }
  lista_nueva→primero = NULL;
  lista_nueva→ultimo = NULL;
  lista_nueva→largo = 0;
  return lista_nueva;
}


void printList(lista_t* lista){
  //printf("Imprimiendo la lista\n");
  nodo_t *nodo_actual = lista→primero;
  while (nodo_actual ≠ NULL){
    float dato = obtener_dato(nodo_actual);
    printf("%f ",dato);
    nodo_actual = obtener_referencia(nodo_actual);
  }
  printf("\n");
}


void buscar(nodo_t* nodo_ant, nodo_t* nodo_sig,float dato){
  //printf("en buscar con:%f ",dato);
  if (nodo_sig ≡ NULL){
    //printf("llegue al final ");
    float dato_act = obtener_dato(nodo_ant);
    if (dato ≤ dato_act){
      set_dato_nodo(nodo_ant,dato);
      //printf("dato_act:%f\n",dato_act);
      definir_referencia(nodo_ant,crear_nodo(dato_act,NULL));
      //printf("dato del nodo sa:%f\n",obtener_dato(nodo_ant));
    }else{
      definir_referencia(nodo_ant,crear_nodo(dato,NULL));
    }
  }else{
    //printf("not null\n");
    float num = obtener_dato(nodo_ant);
    if (dato ≤ num){
      set_dato_nodo(nodo_ant,dato);
      nodo_t* nuevo = crear_nodo(num,nodo_sig);
      definir_referencia(nodo_ant,nuevo);
    }else{
      buscar(nodo_sig,obtener_referencia(nodo_sig),dato);
    }
  }
}

int getLargo(lista_t* lista){
```

sep 06, 16 18:42 **lista.c** Page 2/2

```
67    return lista→largo;
68  }
69
70  bool lista_insertar(lista_t *lista, float dato){
71    //printf("insertando el elemento:%.2f\n",dato);
72    if (lista→largo ≡ 0){
73      //printf("lista vacia ");
74      nodo_t *nuevo_nodo = crear_nodo(dato,NULL);
75      if (nuevo_nodo ≠ NULL){
76        lista→primero = nuevo_nodo;
77      }
78    }else{
79      buscar(lista→primero,obtener_referencia(lista→primero),dato);
80    }
81    lista→largo++;
82    return true;
83  }
84
85  bool lista_esta_vacia(const lista_t *lista){
86    return lista→primero ≡ NULL;
87  }
88
89  //devuelvo -1 o la posicion donde se encuentra el dato
90  float lista_posicion(lista_t *lista,int posicion){
91    nodo_t *nodo_actual = lista→primero;
92    int posActual = 0;
93    while (nodo_actual ≠ NULL){
94      if (posicion ≡ posActual){
95        return obtener_dato(nodo_actual);
96      }else{
97        nodo_actual = obtener_referencia(nodo_actual);
98        posActual++;
99      }
100   }
101   return 0;
102   }
103
104
105
106 void lista_destruir(lista_t *lista){
107   while (¬lista_esta_vacia(lista)){
108     lista_borrar_primero(lista);
109   }
110   free(lista);
111 }
112
113
114 void lista_borrar_primero(lista_t *lista){
115   if (lista_esta_vacia(lista) ≡ false){
116     nodo_t* primero = lista→primero;
117     //int dato_auxiliar = obtener_dato(primero);
118     lista→primero = obtener_referencia(primero);
119     if (lista→primero≡NULL){
120       lista→ultimo = NULL;
121       }
122     destruir_nodo(primero);
123     lista→largo--;
124   }
125 }
```

sep 06, 16 18:42 **file.h** Page 1/1

```
1   #ifndef FILE_H
2   #define FILE_H
3
4
5   typedef struct file file_t;
6
7   //Funciones
8   file_t* file_open(char *direccion, char *modo);
9   int file_read(file_t *archivo, void* buffer);
10  int file_end(file_t *archivo);
11  int file_close(file_t *archivo);
12
13
14  #endif
```

sep 06, 16 18:42 **file.c** Page 1/1

```c
#include "file.h"
#include <stdlib.h> //malloc()
#include <stdio.h> //funciones de file

struct file{
  FILE* fd; //file descriptor
};

file_t* file_open(char *direccion, char *modo){
  file_t *file = malloc(sizeof(file_t));
  FILE* file_aux = fopen(direccion, modo);
  if (file_aux == NULL){
    printf("Is NULL");
    return NULL;
  }
  file->fd = file_aux;
  return file;
}

//cada vez que llamo a la funciÃ³n voy leyendo 5 bytes es decir una mediciÃ³n
int file_read(file_t *archivo, void* buffer){
  int num = (int)fread(buffer, sizeof(short int), 1, archivo->fd);
  return num;
}

int file_end(file_t *archivo){
  return feof(archivo->fd);
}


int file_close(file_t *archivo){
  fclose(archivo->fd);
  free(archivo);
  return 0;
}
```

sep 06, 16 18:42 **conectador.h** Page 1/1

```c
#ifndef CONECTADOR_H
#define CONECTADOR_H


typedef struct conectador conectador_t;


conectador_t* socket_conectador_init(int descriptor);
conectador_t* socket_conectador_create();
int socket_conectador_send(conectador_t *conectador, void *buffer,int longitud);
int socket_conectador_receive(conectador_t *conect, void *buffer,int longitud);
int socket_conectador_close(conectador_t *conectador);
int socket_conectador_connect(conectador_t *conectador, char *ip, int port);

#endif
```

```c
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include "conectador.h"
#include <stdlib.h>
#include <string.h>

struct conectador{
  int descriptor_file;
};


conectador_t* socket_conectador_init(int descriptor){
  conectador_t* conectador = malloc(sizeof(struct conectador));
  if (conectador ≠ NULL) {
    conectador→descriptor_file = descriptor;
  }
  return conectador;
}

conectador_t* socket_conectador_create(){
  int descriptor_file_aux;
  descriptor_file_aux = socket(AF_INET,SOCK_STREAM,0);
  if (descriptor_file_aux ≡ −1) {
    printf("Error crear socket\n");
    return NULL;
  }

  conectador_t* conectador = malloc(sizeof(struct conectador));
  if (conectador ≠ NULL) {
    conectador→descriptor_file = descriptor_file_aux;
  }
  return conectador;
}



int socket_conectador_send(conectador_t *conectador, void *buffer,int longitud){
  int aux = 0; // Guardaremos el valor devuelto por send() */
  int leido = 0; // NÃºmero de caracteres leÃ-dos hasta el momento
  //aux es la cantidad de bytes que envie
  //longitud es la longitud del buffer
  while (leido < longitud){
    aux = send(conectador→descriptor_file, buffer,longitud−leido ,MSG_NOSIGNAL)
;
    if (aux > 0) {
      leido += aux;
    }else {
      if (aux < 0) {
        printf("Error al escribir\n");
        return −1; //por ahora mando sÃ³lo −1, despuÃ©s voy a identicar los erro
res
      }else{
        printf("Socket cerrado\n");
      }
    }
  }
  return 1;
}

int socket_conectador_receive(conectador_t *conect, void *buffer,int longitud){
  int aux = 0; // Guardaremos el valor devuelto por send() */
  int leido = 0; // NÃºmero de caracteres leÃ-dos hasta el momento
  //aux es la cantidad de bytes que envie
```



```c
  //longitud es la longitud del buffer
  while (leido < longitud){
    aux = recv(conect→descriptor_file, buffer,longitud−leido, MSG_NOSIGNAL);
    if (aux > 0) {
      leido += aux;
    }else{
      if (aux < 0) {
        printf("Error al leer\n");
        return −1; //por ahora mando sÃ³lo −1, despuÃ©s voy a identicar los erro
res
      }else{
        //printf("Socket cerrado y envio −1\n");
        return −1;
      }
    }
  }
  return 1;
}


int socket_conectador_connect(conectador_t *conect, char *ip, int port){
  struct sockaddr_in addr;
  addr.sin_port = (unsigned long)port;
  inet_pton(AF_INET, ip , &(addr.sin_addr));
  addr.sin_family = AF_INET;
  int aux;

  aux = connect(conect→descriptor_file, (struct sockaddr*)&addr, sizeof(addr));
  if (aux ≡ −1){
    return −1;
  }
  return 1;
}


int socket_conectador_close(conectador_t *conectador){
  shutdown(conectador→descriptor_file, SHUT_RDWR);
  close(conectador→descriptor_file);
  free(conectador);
  return 1;
}
```

sep 06, 16 18:42 **client.h** Page 1/1

```c
#ifndef CLIENT_H
#define CLIENT_H


#include <stdio.h> //printf()
#include <netinet/in.h> //htons()
#include <stdlib.h> //atoi()
#include <string.h> //srtlen()

#include "time.h"
#include "client.h"
#include "file.h"
#include "calculateTemp.h"
#include "conectador.h"


//Funciones

int client(int argc,char* argv[]);

#endif
```

sep 06, 16 18:42 **client.c** Page 1/2

```c
#include "client.h"


conectador_t* client_create(char *ip,char *puerto){
  //printf("cliente_create\n");
  //printf("ip:%s port:%s\n",ip,puerto);
  conectador_t *self = socket_conectador_create();
  int port = atoi(puerto);
  //printf("el int del puerto es:%d\n",port);
  int aux = socket_conectador_connect(self,ip,port);
  if (aux == -1){
    printf("Error al conectar con el server\n");
    socket_conectador_close(self);
    return NULL;
  }
  return self;
}

int getQuantity(int segundos, float velocidad){
  int diferencia = 60 - segundos;
  float cantidad = velocidad*diferencia;
  return (int)(cantidad + 0.5);
}

int obtenterTemperatura(file_t* file, char temperatura[5], int largo){
  char hexa[5] = "";
  unsigned short int buffer = 0;
  int cantidadBytes = file_read(file,&buffer);
  buffer = htons(buffer);
  snprintf(hexa,sizeof(hexa),"%04X",buffer);
  calcular(hexa,temperatura,largo);
  return cantidadBytes;
}

//esta funcion se encarga de enviar todos los datos respecto al tiempo
int send_time(conectador_t* conectador,int (*list)[6],char* time_char,int cant){
  snprintf(time_char,cant,"%d.%02d.%02d-%02d:%02d:00",
  (*list)[0],(*list)[1],(*list)[2],(*list)[3],(*list)[4]);
  //printf("time_char:%s\n",time_char);
  socket_conectador_send(conectador,time_char,cant);
  return 0;
}

int client_free_memory(file_t* file, conectador_t *conect,char* date){
  socket_conectador_close(conect);
  file_close(file);
  free(date);
  return 0;
}


//-------------------------FUNCION PRINCIPAL-----------------------------

//tp client ip port id frencuencia time file
//   1      2   3    4    5        6   7
int client(int argc, char* argv[]){
  int cantArguments = 8;
  int timeArray[6]; //aca se guarda la hora,minuto y segundo
  char charTemperatura[6]; //aca se guarda el temperatura
  //comprobar cant parametros
  if (cantArguments != argc){
    printf("Tengo %d argumentos y espero %d argumentos\n",argc,cantArguments);
    return 1;
  }
  //printf("Se entrÃ³ correctamente a client y la cantidad de parametros ok\n");

```

sep 06, 16 18:42 **client.c** Page 2/2

```c
    // conectandome con el servidor
    conectador_t* conectador = client_create(argv[2],argv[3]);
    //int largo = (int)strlen(argv[4]);
    //socket_acept_accept
    //envio mi nombre
    socket_conectador_send(conectador,argv[4],7);

    //Lectura
    file_t* file = file_open(argv[7],"rb");

    //la velocidad es mediciones por segundo
    int frecuencia = atoi(argv[5]);
    float velocidad = calcularVelocidad(frecuencia);

    //parseo la hora y lo guardo en el array
    getHrMinSec(argv[6],&timeArray);

    //variables para el envio la fecha y hora
    int long_format_time = 20;
    char* time_char = malloc(sizeof(char)*long_format_time);

    int char_tem_long = 6;
    int cantidad;
    int n = 0; //es el contador de datos que se enviaron
    char espacio[] = " "; //se utiliza para indicar que hay datos por enviar
    char barraN[] = "\n"; //se utiliza para indicar que ya no hay datos
    int long_identificador = 1;
    int hayMediciones; //es para indicar si hay mediciones disponibles

    printTime(&timeArray);
    //envio el date
    send_time(conectador,&timeArray,time_char,long_format_time);
    //calculo cantidad de datos a enviar
    cantidad = getQuantity(timeArray[5],velocidad);
    hayMediciones = obtenterTemperatura(file,charTemperatura,char_tem_long);
    while (hayMediciones){
      //fprintf(stderr,"%s",charTemperatura);
      socket_conectador_send(conectador,charTemperatura,char_tem_long);
      n++;
      //si ya envie la cantidad correspondiente
      hayMediciones = obtenterTemperatura(file,charTemperatura,char_tem_long);
      if (n ≡ cantidad ∨ ¬hayMediciones){
        socket_conectador_send(conectador,barraN,long_identificador);
        //fprintf(stderr,"%s",barraN);
        fprintf(stderr,"Enviando %d muestras\n",n);
        wait(&timeArray);
        if (hayMediciones){
          printTime(&timeArray); //esto es para el proximo envio
          send_time(conectador,&timeArray,time_char,long_format_time);
        }
        n = 0; //reinicio el contador;
        cantidad = getQuantity(timeArray[5],velocidad);
      }else{
        //fprintf(stderr,"%s",espacio);
        socket_conectador_send(conectador,espacio,long_identificador);
      }
    }
    socket_conectador_send(conectador,barraN,long_identificador);
    //printf("Enviando %d muestras\n",n);

    //cierro conexion
    client_free_memory(file,conectador,time_char);


    return 0;
}
```

sep 06, 16 18:42 **calculateTemp.h** Page 1/1

```c
#ifndef CALCULATETEMP_H
#define CALCULATETEMP_H



//Funciones

int calcular(char hexa[5],char temperatura[5],int largo);
float calcularVelocidad(int frecuencia);

#endif
```

sep 06, 16 18:42 **calculateTemp.c** Page 1/1

```c
#include <stdio.h> //printf()
#include <stdlib.h> //atoi()
#include <math.h> //pow()
#include <string.h>

//Este modulo se encarga de todo lo que respecta al calcula y validaciÃ³n de
//un temperatura recibida en notaciÃ³n hexadecimal



float calcularVelocidad(int frecuencia){
  //printf("La cantidad rencuencia es:%d\n",frecuencia);
  //printf("La velocidad es:%d\n",velocidad);
  return 1000.0/frecuencia;
  }


//calcula de manera lineal la temperatura
float tranformar(int temp){
  float tempf = (float)temp;
  //printf("conversion:%f\n",tempf);
  return (0.1)*tempf - 17.0;
}

int conversionHexDec(char* hexadecimal) {
    //printf("Hexadecimal:%s ",hexadecimal);
    int numDecimal = 0;
    char hexDigitos[16]={'0','1','2','3','4','5','6','7','8','9','A','B','C','D'
    ,'E','F'};
    int i, j;
    int potencia = 0; // potencia a la que se eleva el numero 16

    /* Converting hexadecimal number to decimal number */
    for (i = strlen(hexadecimal)-1; i ≥ 0; i--) {
        for (j = 0; j < 16; j++){
            if (hexadecimal[i] ≡ hexDigitos[j]){
                numDecimal += j*pow(16, potencia);
            }
        }
        potencia++;
    }
    //printf("Decimal:%d ", numDecimal);
    return numDecimal;
}

float validarTemperatura(float anterior, float actual){
  if ((actual > -17.00) ∧  (actual < 59.70)){
    return actual;
  }else{
    return anterior;
  }
}



int calcular(char hexa[5],char temp[5],int largo){
  int num = conversionHexDec(hexa);
  //printf("El num entero:%d ",num);
  float numTrans = tranformar(num);
  //printf("float:%f\n", numTrans);
  snprintf(temp,largo,"%.2f",numTrans);
  return 0;
}
```

sep 06, 16 18:42 **aceptador.h** Page 1/1

```c
#ifndef ACEPTADOR_H
#define ACEPTADOR_H


#include "conectador.h"

typedef struct aceptador aceptador_t;
//typedef struct conectador conectador_t;

aceptador_t* socket_acept_create();
int socket_acept_connect(aceptador_t *aceptador, int port, int cant, char *ip);
int socket_acept_accept(aceptador_t *aceptador, conectador_t ** conectador);
int socket_acept_close(aceptador_t *aceptador);

#endif
```

```
1  #include "aceptador.h"
2
3  #include <stdio.h>
4  #include <string.h>
5  #include <sys/socket.h>
6  #include <netinet/in.h>
7  #include <arpa/inet.h>
8  #include <unistd.h>
9  #include <stdlib.h>
10 #include <errno.h>
11 struct aceptador{
12   int descriptor;
13 };
14
15
16 //-----------------------------------------------
17
18
19 aceptador_t* socket_acept_create(){
20   int aux = socket(AF_INET,SOCK_STREAM,0);
21   if (aux ≡ −1) {
22     printf("Error en crear socket aceptador");
23     return NULL;
24   }
25   aceptador_t *aceptador = malloc(sizeof(aceptador_t));
26   aceptador→descriptor = aux;
27   return aceptador;
28 }
29
30
31
32 int socket_acept_connect(aceptador_t *acept, int port, int cant, char *ip){
33   struct sockaddr_in address;
34   inet_pton(AF_INET,"127.0.0.1", &(address.sin_addr));
35   address.sin_port = (unsigned long)port;
36   address.sin_family = AF_INET;
37
38   int yes=1;
39   setsockopt(acept→descriptor,SOL_SOCKET,SO_REUSEADDR,&yes,sizeof(int));
40   //aviso al sistema que asocie mi programa con el socket que abrÃ-
41   if (bind(acept→descriptor,(struct sockaddr *)&address,sizeof(address))≡−1){
42     printf("%d\n",errno);
43     strerror(errno);
44     printf("Error en bind()\n");
45     return −1;
46   }
47   // aviso al sistema que ya puede empezar a reservar clientes, los encola
48   if(listen(acept→descriptor,cant) ≡ −1){
49     printf("Error en listen()\n");
50     return −1;
51   }
52   return 1;
53 }
54
55
56 int socket_acept_accept(aceptador_t *aceptador,conectador_t **conectador){
57   int descriptor_comunicador = accept(aceptador→descriptor,NULL,NULL);
58   if(descriptor_comunicador ≡ −1){
59     printf("Error en accept()\n");
60   }
61   *conectador = socket_conectador_init(descriptor_comunicador);
62   return 1;
63 }
64
65
66 int socket_acept_close(aceptador_t *aceptador){
```



```
67   shutdown(aceptador→descriptor,SHUT_RDWR);
68   close(aceptador→descriptor);
69   free(aceptador);
70   return 1;
71 }
```

sep 06, 16 18:42 **Table of Content** Page 1/1

Table of Contents

1	*time.h*	sheets	1	to	1	(1) pages	1− 1	13	lines
2	*time.c*	sheets	1	to	2	(2) pages	2− 3	76	lines
3	*server.h*	sheets	2	to	2	(1) pages	4− 4	19	lines
4	*server.c*	sheets	3	to	4	(2) pages	5− 7	163	lines
5	*principal.c*	sheets	4	to	4	(1) pages	8− 8	37	lines
6	*nodo.h*	sheets	5	to	5	(1) pages	9− 9	21	lines
7	*nodo.c*	sheets	5	to	5	(1) pages	10− 10	47	lines
8	*lista.h*	sheets	6	to	6	(1) pages	11− 11	29	lines
9	*lista.c*	sheets	6	to	7	(2) pages	12− 13	126	lines
10	*file.h*	sheets	7	to	7	(1) pages	14− 14	15	lines
11	*file.c*	sheets	8	to	8	(1) pages	15− 15	36	lines
12	*conectador.h*	sheets	8	to	8	(1) pages	16− 16	16	lines
13	*conectador.c*	sheets	9	to	9	(1) pages	17− 18	105	lines
14	*client.h*	sheets	10	to	10	(1) pages	19− 19	22	lines
15	*client.c*	sheets	10	to	11	(2) pages	20− 21	133	lines
16	*calculateTemp.h*	sheets	11	to	11	(1) pages	22− 22	12	lines
17	*calculateTemp.c*	sheets	12	to	12	(1) pages	23− 23	64	lines
18	*aceptador.h*	sheets	12	to	12	(1) pages	24− 24	16	lines
19	*aceptador.c*	sheets	13	to	13	(1) pages	25− 26	72	lines