```c
#define LOCALHOST "127.0.0.1"
#define WEBPORT "80"

// Mensajes de error
#define ERROR_ABRIR_SOCKET "Error al abrir el socket \n"
#define ERROR_BIND_SOCKET "Error al ligar el socket \n"
#define ERROR_ACCEPT_SOCKET "Error al aceptar la conexion \n"
#define ERROR_RECV_SOCKET "Error al leer del socket \n"
#define ERROR_SEND_SOCKET "Error al escribir en el socket \n"
#define ERROR_FORK "Error al hacer fork \n"
#define ERROR_ABRIR_ARCHIVO "Error al abrir el archivo \n"
#define ERROR_PIPE "Error al hacer el pipe \n"
#define ERROR_DAEMON "Error al poner el programa en background \n"
#define ERROR_UNEXPECTED_END "El servidor finalizo de manera inesperada \n"
#define ERROR_IP_PORT "La IP o el puerto ingresado es invalido \n"
#define ERROR_INPUT_DATOS "Error en el ingreso de datos \n"

// Mensajes de respuesta HTTP https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html
#define RTA_200 "HTTP/1.0 200 OK \n"
#define RTA_400 "HTTP/1.0 400 Bad Request \n"
#define RTA_403 "HTTP/1.0 403 Forbidden \n"
#define RTA_404 "HTTP/1.0 404 Not Found \n"
#define RTA_501 "HTTP/1.0 501 Not Implemented \n"

// Tipos de Contenido usados en el proyecto
#define CT_HTML "Content-type: text/html \n\n"
#define CT_JPG "Content-type: image/jpeg \n\n"
#define CT_PNG "Content-type: image/png \n\n"
#define CT_GIF "Content-type: image/gif \n\n"

#ifndef SERVIDORHTTP_H_   /* Include guard */
#define SERVIDORHTTP_H_

/** error:
 * Muestra el mensaje de error y termina el programa con EXIT_FAILURE
 * DE:     Mensaje (string)
 * */
void error(char *msg);

/** log_error:
 * Registra el error dado en el system log de servidor HTTP.
 * DE:     Mensaje (string)
 * */
void log_error(char *msg);

/** log_info:
 * Registra la informacion dada en el system log de servidor HTTP.
 * DE:     Mensaje (string)
 * */
void log_info(char *msg);

/** ayuda:
 * Muestra el mensaje de ayuda al usuario y termina el programa con EXIT_SUCCESS
 * */
void ayuda();

```

```
57 /** signalHandler:
58  * Método que se encarga de manejar las señales recibidas.
59  * DE:     Signal (int), el código de la señal a manejar.
60  * */
61 void signalHandler(int sig);
62
63 /** verificarIP:
64  * Dada una IP, verifica si la IP corresponde a una IPv4 valida.
65  * DE:     Servidor (string), la IP a analizar.
66  * DS:    1 si la IP es valida, 0 en caso contrario.
67  * */
68 int verificarIP(char * servidor);
69
70 /** verificaPuerto:
71  * Dado un puerto, verifica si el puerto es valido (esta entre 1 y 65535).
72  * DE:     Puerto (string), el puerto a analizar..
73  * DS:    1 si el puerto es valido, 0 en caso contrario.
74  * */
75 int verificarPuerto(char * puerto);
76
77 /** parseMsg:
78  * Analiza el mensaje brindado en el buffer y retorna cuál es el tipo de mensa
    je,
79  * la ruta del archivo y el protocolo utilizado. No se encarga de hacer cheque
    os
80  * en cuanto a la correctitud del mensaje.
81  * DE:    Buf, el buffer de entrada que posee el mensaje.
82  * DS:     TipoMsg, Ruta y Protocolo, los datos de salida según el análisis si
    ntáctico.
83  *  */
84 void parseMsg (char * buf, char ** tipoMsg, char ** ruta, char ** protocolo);
85
86 /** archivoExiste:
87  * Dada la ruta de un archivo, retorna verdadero si el archivo existe y falso
    en caso contrario.
88  * DE:     Archivo (string), la ruta del archivo.
89  * DS:     1 si el archivo existe, 0 en caso contrario.
90  * */
91 int archivoExiste(char * archivo);
92
93 /** archivoAbrible:
94  * Dada la ruta de un archivo, retorna verdadero si el archivo se puede abrir
    y falso en caso contrario.
95  * DE:     Archivo (string), la ruta del archivo.
96  * DS:     1 si el archivo se puede abrir, 0 en caso contrario.
97  * */
98 int archivoAbrible(char * archivo);
99
100 /** archivoSize:
101  * Dada la ruta de un archivo, retorna el tamaño del contenido del archivo.
102  * DE:     Archivo (string), la ruta del archivo.
103  * DS:     Size (long), el tamaño del contenido del archivo.
104  * */
105 long archivoSize(char * archivo);
106
107 /** esGet:
108  * Dada una cadena de texto, analiza si la misma es un "GET"
109  * DE:     Mensaje (string), la cadena de texto.
```

```
110  * DS:     1 si la cadena es "GET", 0 en caso contrario.
111  * */
112 int esGet(char * msg);
113 
114 /** esBarra:
115  * Dada una cadena de texto, analiza si la misma es una barra "/"
116  * DE:     Mensaje (string), la cadena de texto.
117  * DS:     1 si la cadena es una barra "/", 0 en caso contrario.
118  * */
119 int esBarra(char * msg);
120 
121 /** mandarHeader:
122  * Dada una cadena de texto y un socket, envia el texto a traves de ese socket
    .
123  * DE:     Sock (int), el socket asociado por donde mandar el mensaje.
124  *         Resp (string), el mensaje a ser enviado a traves del socket.
125  * */
126 void mandarHeader(int sock, char * resp);
127 
128 /** mandarHeaders:
129  * Método para encapsular y mandar 2 mensajes a traves de un socket.
130  * Hace dos llamadas a el metodo "mandarHeader", con tipoResp y con tipoCont.
131  * Normalmente se usa para mandar un tipo de respuesta (status)
132  * y el tipo de contenido a ser enviado (content type).
133  * DE:     Sock (int), el socket asociado por donde mandar el mensaje.
134  *         TipoResp (string), el tipo de respuesta a enviar.
135  *         TipoCont (string), el tipo de contenido a enviar.
136  * */
137 void mandarHeaders(int sock, char * tipoResp, char * tipoCont);
138 
139 /** mandarRechazo:
140  * Dado un socket, un tipo de respuesta, un titulo y un mensaje,
141  * el método se encarga de mandar una respuesta del estado de tipoResp.
142  * Y además se encarga de crear una página de contenido HTML con el
143  * título y el mensaje otorgado. El método normalmente se usa para mandar
144  * respuestas de tipo 4XX o 5XX y mostrar de forma amigable una página,
145  * para los que esten haciendo solicitudes mediante un navegador web.
146  * DE:     Sock (int), el socket asociado por donde mandar el mensaje.
147  *         TipoResp (string), el tipo de respuesta a ser enviado.
148  *         Titulo (string), título a ser mostrado en el contenido HTML.
149  *         Mensaje (string), mensaje a ser mostrado en el contenido HTML.
150  * */
151 void mandarRechazo(int sock, char * tipoResp, char * titulo, char * mensaje);
152 
153 /** mandarArchivo:
154  * Dado un socket y la ruta de un archivo, manda el archivo
155  * en modo binario a traves del socket.
156  * DE:     Socket (int), el socket asociado para mandar el archivo.
157  *         Archivo (string), la ruta del archivo.
158  * */
159 void mandarArchivo(int sock, char * archivo);
160 
161 /** esHTML:
162  * Dada una cadena de texto referente a la ubicación de un archivo,
163  * revisa si el archivo tiene la extension HTML o HTM.
164  * DE:     Archivo (string), la ruta del archivo.
165  * DS:     1 si el archivo es HTML/HTM, 0 en caso contrario.
166  * */
```

```
167 int esHTML(char * archivo);
168
169 /** esJPG:
170  * Dada una cadena de texto referente a la ubicación de un archivo,
171  * revisa si el archivo tiene la extension JPG.
172  * DE:     Archivo (string), la ruta del archivo.
173  * DS:     1 si el archivo es JPG, 0 en caso contrario.
174  * */
175 int esJPG(char * archivo);
176
177 /** esGIF:
178  * Dada una cadena de texto referente a la ubicación de un archivo,
179  * revisa si el archivo tiene la extension GIF.
180  * DE:     Archivo (string), la ruta del archivo.
181  * DS:     1 si el archivo es GIF, 0 en caso contrario.
182  * */
183 int esGIF(char * archivo);
184
185 /** esPNG:
186  * Dada una cadena de texto referente a la ubicación de un archivo,
187  * revisa si el archivo tiene la extension PNG.
188  * DE:     Archivo (string), la ruta del archivo.
189  * DS:     1 si el archivo es PNG, 0 en caso contrario.
190  * */
191 int esPNG(char * archivo);
192
193 /** esPHP:
194  * Dada una cadena de texto referente a la ubicación de un archivo,
195  * revisa si el archivo tiene la extension PHP.
196  * DE:     Archivo (string), la ruta del archivo.
197  * DS:     1 si el archivo es PHP, 0 en caso contrario.
198  * */
199 int esPHP(char * archivo);
200
201 /** getExtension:
202  * Dada una cadena de texto, obtiene su extension (si la tuviese).
203  * DE:     Archivo (string), la ruta del archivo para obtener su extensión.
204  * DS:    Retorna un puntero a un string con la extensión, o NULL en caso de n
    o
205  *         poseer una extensión reconocible.
206  * */
207 char * getExtension(char* archivo);
208
209 /** appchr:
210  * Dada una cadena de texto y un caracter, agrega dicho caracter
211  * al final de la cadena de texto (append).
212  * DE:     Str (string), la cadena de texto.
213  *         Chr (char), la letra para agregar al final de la cadena.
214  * DS:    Retorna un puntero a la primera posición de la cadena de texto resul
    tante.
215  * */
216 char * appchr(char * str, const char chr);
217
218 /** recibirMensaje:
219  * Dado un socket, se hace una lectura a través de dicho socket hasta que
220  * se lean dos enter seguidos (CR-LF o LF-CR).
221  * Luego, se retorna un string con el contenido de lo leído.
222  * DE:     Socket (int), el socket asociado para hacer la lectura.
```

```
223  * DS:    Retorna un puntero a la primera posición de la cadena de texto leída
     .
224  * */
225 char * recibirMensaje(int sock);
226 
227 /** minusculas:
228  * Dada una cadena de caracteres, devuelve la misma cadena pero convertida a m
     inúsculas.
229  * DE:     Str (string), la cadena de caracteres para convertir.
230  * DS:    Retorna un puntero a la primera posición de la cadena de texto en mi
     núsculas.
231  * */
232 char * minusculas(char * str);
233 
234  /** verificarPHP:
235  * Método para desglosar una ruta seguida de argumentos. Se utiliza para los a
     rchivos
236  * PHP que reciben parámetros, para poder separar la ruta del archivo de sus a
     rgumentos.
237  * El método retorna en su primer dato únicamente la ruta del archivo, y en su
      segundo argumento
238  * los parámetros dados, si los hubiera.
239  * DE:     Archivo (string), la ruta del archivo a desglosar.
240  * DS:    Archivo (string), la ruta del archivo, separado de sus parametros.
241  *         Argumentos (string), los parametros separados de la ruta del archiv
     o, si existiesen.
242  *         En caso de no existir parámetros, argumentos será NULL.
243  * */
244 void verificarPHP(char ** archivo, char ** argumentos);
245 
246 /** procesarPHP:
247  * Método para atender el pedido a un archivo PHP. Dada la ruta de un archivo
     y
248  * sus respectivos parámetros, el método se encarga de llamar al CGI de PHP
249  * y de responder a través del socket según lo resultante de PHP-CGI.
250  * DE:     Socket (int), el socket asociado al hilo donde se responderá.
251  *         Archivo (string), la ruta del archivo PHP.
252  *         Parametros (string), los parámetros de la ejecución, si existiesen.
253  * */
254 void procesarPHP(int sock, char * archivo, char * parametros);
255 
256 /** atenderPedido:
257  * Método principal. Dado un socket, se encarga de atenderlo.
258  * El método se encarga de toda la inteligencia del servidor:
259  * - Obtener el mensaje, analizarlo y devolver el Response HTTP según correspo
     nda.
260  * DE:     Socket (int), el socket asociado al hilo donde se harán las lectura
     s y escrituras.
261  * */
262 void atenderPedido(int sock);
263 
264 #endif // SERVIDORHTTP_H_
265 
```