Your code must comply with the C++ 98 standard and should still compile when adding the flag -std=c++98.

Allowed external functs :
All functionality must be implemented in C++ 98.
execve, pipe, strerror, gai_strerror, errno, dup, dup2, fork, socketpair, htons, htonl, ntohs, ntohl, select, poll, epoll (epoll_create, epoll_ctl, epoll_wait), kqueue (kqueue, kevent), socket, accept, listen, send, recv, chdir, bind, connect, getaddrinfo, freeaddrinfo, setsockopt, getsockname, getprotobyname, fcntl, close, read, write, waitpid, kill, signal, access, stat, open, opendir, readdir and closedir.

You must write an HTTP server in C++ 98.
Your executable should be executed as follows:
./webserv [configuration file]

1 Requirements
• Your program must use a configuration file, provided as an argument on the command line, or available in a default path.
• You cannot execve another web server.
• Your server must remain non-blocking at all times and properly handle client disconnections when necessary.
• It must be non-blocking and use only 1 poll() (or equivalent) for all the I/O operations between the clients and the server (listen included).
• poll() (or equivalent) must monitor both reading and writing simultaneously.
• You must never do a read or a write operation without going through poll() (or equivalent).
• Checking the value of errno to adjust the server behaviour is strictly forbidden after performing a read or write operation.
• You are not required to use poll() (or equivalent) before read() to retrieve your configuration file.
• When using poll() or any equivalent call, you can use every associated macro or helper function (e.g., FD_SET for select()).
• A request to your server should never hang indefinitely.
• Your server must be compatible with standard web browsers of your choice.
• NGINX may be used to compare headers and answer behaviours (pay attention to differences between HTTP versions).
• Your HTTP response status codes must be accurate.
• Your server must have default error pages if none are provided.
• You can’t use fork for anything other than CGI (like PHP, or Python, and so forth).
• You must be able to serve a fully static website.
• Clients must be able to upload files.
• You need at least the GET, POST, and DELETE methods.
• Stress test your server to ensure it remains available at all times.
• Your server must be able to listen to multiple ports to deliver different content (see Configuration file).

2 Configuration file
In the configuration file, you should be able to:

• Define all the interface:port pairs on which your server will listen to (defining multiple websites served by your program).
• Set up default error pages.
• Set the maximum allowed size for client request bodies.
• Specify rules or configurations on a URL/route (no regex required here), for a website, among the following:
◦ List of accepted HTTP methods for the route.
◦ HTTP redirection.
◦ Directory where the requested file should be located (e.g., if URL /kapouet is rooted to /tmp/www, URL /kapouet/pouic/toto/pouet will search for /tmp/www/pouic/toto/pouet).
◦ Enabling or disabling directory listing.
◦ Default file to serve when the requested resource is a directory.
◦ Uploading files from the clients to the server is authorized, and storage location is provided.
◦ Execution of CGI, based on file extension (for example .php). Here are some specific remarks regarding CGIs:
* Do you wonder what a CGI is?
* Have a careful look at the environment variables involved in the web server-CGI communication. The full request and arguments provided by the client must be available to the CGI.
* Just remember that, for chunked requests, your server needs to un-chunk them, the CGI will expect EOF as the end of the body.
* The same applies to the output of the CGI. If no content_length is returned from the CGI, EOF will mark the end of the returned data.
* The CGI should be run in the correct directory for relative path file access.
* Your server should support at least one CGI (php-CGI, Python, and so forth).

You must provide configuration files and default files to test and demonstrate that every feature works during the evaluation.
You can have other rules or configuration information in your file (e.g., a server name for a website if you plan to implement virtual hosts).