```
#include <netinet/in.h>

struct sockaddr_in {
    short            sin_family;   // e.g. AF_INET
    unsigned short   sin_port;     // e.g. htons(3490)
    struct in_addr   sin_addr;     // see struct in_addr, below
    char             sin_zero[8];  // zero this if you want to
};

struct in_addr {
    unsigned long s_addr;  // load with inet_aton()
};


struct sockaddr_in myaddr;
int s;

myaddr.sin_family = AF_INET;
myaddr.sin_port = htons(3490);
inet_aton("63.161.169.137", &myaddr.sin_addr.s_addr);

s = socket(PF_INET, SOCK_STREAM, 0);


//struct sockaddr_in self;
//bzero ((char *) &self, sizeof(self));
//self.sin_family = AF_INET;
//self.sin_port = htons(port);
//self.sin_addr.s_addr = INADDR_ANY;
```

The function **inet_ntoa()** converts a network address in a struct in_addr to a dots-and-numbers format string. The "n" in "ntoa" stands for network, and the "a" stands for ASCII for historical reasons (so it's "Network To ASCII"--the "toa" suffix has an analogous friend in the C library called atoi() which converts an ASCII string to an integer.)

The function **inet_aton()** is the opposite, converting from a dots-and-numbers string into a in_addr_t (which is the type of the field s_addr in your struct in_addr.)

The **htons()** function converts the unsigned short integer hostshort from host byte order to network byte order.

```
int bind(int sockfd, struct sockaddr *my_addr, socklen_t addrlen);
//bind(SocketD,(struct sockaddr*)&self,sizeof(self))
```

```
int listen(int s, int backlog);
//listen(SocketD,20);

int accept(int s, struct sockaddr *addr, socklen_t *addrlen);
//struct sockaddr_in client_addr;
//int addrlen = sizeof(client_addr);
//int clientfd = accept(SocketD,(struct sockaddr*)&client_addr,(socklen_t*)&addrlen);
```

```
typedef struct linger {
  u_short l_onoff;
  u_short l_linger;
} LINGER, *PLINGER, *LPLINGER;
```

Specifies whether a socket should remain open for a specified amount of time after a closesocket function call to enable queued data to be sent. This member can have one of the following values.

0 do not linger DO not remain open  any other value remain open specified time.

```
ssize_t recv(int sockfd, void *buf, size_t len, int flags);
//received = recv(clientfd, rebuffer, 80, 0);

ssize_t send(int sockfd, const void *buf, size_t len, int flags);
//send(clientfd,Hello.c_str(),Hello.length(),0);
```