```
/*

This client will send a string to a server process on the same machine or
a different host.

The server will echo the string back in reverse order.

Compile with: cc UDP_revEcho_client.c –o sendit

Usage: ./sendit server port string

*/

#include<stdlib.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<netdb.h>
#include<stdio.h>
#include<string.h>
#include<errno.h>

#define BUF_LEN 48

main(int argc, char *argv[])
{
        int csd;                        /* client socket descriptor             */
        struct sockaddr_in server;      /* server address structure             */
        struct hostent *server_host;/* pointer to server host details
                                        structure returned by resolver          */
        int server_len;                 /* size of above structure              */
        int string_size;                /* size of send string including
                                           trailing nul                         */
        short server_port;              /* servers port number                  */
        int out_cnt, in_cnt;            /* byte counts for send and receive     */
        char client_send_string[BUF_LEN];
                                        /* buffer to hold send string           */
        char server_reversed_string[BUF_LEN];
                                        /* buffer to hold recieve string        */


        /* Check for correct command line usage */
        if(argc!=4)
        {
                fprintf(stderr,"Usage: %s Server Port send_string\n",argv[0]);
                exit(EXIT_FAILURE);
        }

        /* Grab the command line arguments and decode them */

        /* Use the resolver to get the addresss of the server */
        server_host=gethostbyname(argv[1]);
        /* if there's a problem, report it and exit */
        if (server_host == NULL)
        {
                herror("While calling gethostbyname()");
                exit(EXIT_FAILURE);
        }

        server_port=atoi(argv[2]);
        strcpy(client_send_string,argv[3]);
```



```
        /* create the socket */
        csd=socket(PF_INET, SOCK_DGRAM, 0);
        /* if there's a problem, report it and exit */
        if(csd<0)
        {
                perror("While calling socket()");
                exit(EXIT_FAILURE);
        }

        /* we haven't bound the socket to an address or port
           let the system decide what's best */

        /* set up the server address details in preparation for sending
           the message */
        server.sin_family=AF_INET;
        memcpy(&server.sin_addr.s_addr,server_host->h_addr_list[0],
                server_host->h_length);
        server.sin_port=htons(server_port);

        /* set the length so that the trailing nul gets sent as well */
        string_size=strlen(client_send_string)+1;

        /* send the message off to the server */
        out_cnt=sendto(csd, client_send_string, string_size, 0,
                        (struct sockaddr *)&server,sizeof(server));
                /* the 0 if for flags that we don't use here */

        /* if there's a problem, report it and exit */
        if(out_cnt<0)
        {
                perror("While calling sendto()");
                exit(EXIT_FAILURE);
        }

        fprintf(stderr,"You have sent \"%s\"\n",client_send_string);

        fprintf(stderr,"Have reached recvfrom(), should now block until message receipt\n");

        /* get the response from the server and print it */
        server_len=sizeof(server);
        in_cnt=recvfrom(csd, server_reversed_string, BUF_LEN, 0,
                        (struct sockaddr *)&server,(socklen_t *)&server_len);

        /* if there's a problem, report it and exit */
        if(in_cnt<0)
        {
                perror("While calling recvfrom()");
                exit(EXIT_FAILURE);
        }

        fprintf(stderr,"The server has responded with: \"%s\"\n",server_reversed_string);

        /* close the socket now */
        close(csd);
}
```