```c
//Name: Mitchell Larson
//Course: CE 4961
//Assignment: Project 3

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

struct networkData {
    uint32_t value1;
    char string1[6];
    uint8_t value2;
    uint16_t value3;
} __attribute__((packed));

#define DATAMAX 1000

int main(int argc, char** argv){

    //Define variables needed by application
    int sock;
    struct sockaddr_in server;
    struct hostent *hp;
    struct networkData d1 = {
        .value1 = htonl(8675309),
        .string1 = "Jenny\0",
        .value2 = 241,
        .value3 = htons(57005)
    };

    //Get IP address - Exit with error code if unsuccessful
    if(argc != 3){
        printf("Usage: echoclient <IP Address> <port>\n");
        exit(1);
    }

    //Create a socket - Exit if unsuccessful
    if((sock = socket(PF_INET, SOCK_DGRAM, 0)) < 0){
        perror("Error creating socket");
        exit(1);
    }

    server.sin_family = AF_INET;

    //Get host name and print + exit if there was an error.
    if((hp = gethostbyname(argv[1])) == 0){
        perror("Error getting host");
        exit(1);
    }

    //Copy IP address info into server struct
    memcpy(&server.sin_addr.s_addr, hp->h_addr, hp->h_length);

    unsigned short port;

    //Get port from user - Exit with error code if unsuccessful
    if(sscanf(argv[2], "%hu", &port) != 1){
        perror("Error parsing port");
        exit(1);
    }

    //Add port info to server object
    server.sin_port = htons(port);
```

```
70
71      printf("Ready to send to remote server %s at port %hu\n", hp->h_name, port);
72
73      //Send UDP request to server
74      int size_echoed;
75      int size_sent = sendto(sock, &d1, sizeof(d1), 0, (struct sockaddr*) &server,
        sizeof(server));
76
77      //If there was an error send data, indicate to user and exit.
78      if(size_sent < 0){
79          perror("Error Sending data");
80          exit(1);
81      }
82
83      struct timeval timeout = {5,0};
84
85      fd_set sockedReadSet;
86      FD_ZERO(&sockedReadSet);
87      FD_SET(sock, &sockedReadSet);
88
89      //Setup socket so that a timeout is associated with it. This will allow the busy wait
90      //for receiving data to be exited after reaching a time threshold.
91      if(select(sock+1, &sockedReadSet, 0, 0, &timeout) < 0){
92          perror("error on select");
93          exit(1);
94      }
95
96      char rec_data[DATAMAX];
97      memset(rec_data, 0, DATAMAX);
98
99      //If data received from the server, print it out to the user. If the request takes
100     //too long, timeout and print a message indicating that the server could not send
101     //a response.
102     if(FD_ISSET(sock, &sockedReadSet)){
103         size_echoed = recvfrom(sock, rec_data, DATAMAX, 0, NULL, NULL);
104
105         if(size_echoed < 0){
106             perror("Error receiving data");
107             exit(1);
108         }
109
110         printf("Received message: %s\n", rec_data);
111     }else{
112         printf("Timeout occurred - Server isn\'t listening, the packet was dropped,"
113             "or it refused to respond.\n");
114     }
115
116     //Close socket and release it back to the OS.
117     close(sock);
118
119     return 0;
120 }
```