---------------------------------------------------------------------------------------------------

**QUESTION 1**

---------------------------------------------------------------------------------------------------

**Question 1:**

**Readme**

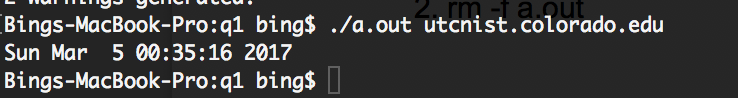
To compile:

1. make

OR

2. gcc main.c

To run(test example): ./a.out utcnist.colorado.edu

The result:

To clean the files:

1. Make clean

OR

2. rm -f a.out

**Make file:**

all:

gcc main.c

clean:

rm -f a.out

**Code:**

#include <sys/types.h>  
#include <unistd.h>  
#include <stdlib.h>  
#include <string.h>  
#include <stdio.h>  
#include <netinet/in.h>  
#include <time.h>  
#include <errno.h>  
#include <netdb.h>  
#include <arpa/inet.h>  
  
#define BUFSIZE 64  
#define UNIXEPOCH 2208988800 /\* UNIX epoch, in UCT secs \*/  
#define MSG "what time is it?\n"  
typedef unsigned long u\_long;  
  
int  
errexit(const char \*format, ...) {  
 va\_list args;  
 vfprintf(stderr, format, args);  
 exit(1);  
}  
  
typedef unsigned short u\_short;  
  
  
/\*------------------------------------------------------------------------  
 \* main - UDP client for TIME service that prints the resulting time  
 \*------------------------------------------------------------------------  
 \*/  
int  
main(int argc, char \*argv[]) {  
 char \*host = "localhost"; /\* host to use if none supplied \*/  
 char \*service = "time"; /\* default service name \*/  
 char \*transport = "udp";  
 time\_t now; /\* 32-bit integer to hold time \*/  
 int s, n, type; /\* socket descriptor, read count\*/  
  
 struct hostent \*phe; /\* pointer to host information entry \*/  
 struct servent \*pse; /\* pointer to service information entry \*/  
 struct protoent \*ppe; /\* pointer to protocol information entry\*/  
 struct sockaddr\_in sin; /\* an Internet endpoint address \*/  
 memset(&sin, 0, sizeof(sin));  
 sin.sin\_family = AF\_INET;  
   
 switch (argc) {  
 case 1:  
 host = "localhost";  
 break;  
 case 3:  
 service = argv[2];  
 /\* FALL THROUGH \*/  
 case 2:  
 host = argv[1];  
 break;  
 default:  
 fprintf(stderr, "usage: UDPtime [host [port]]\n");  
 exit(1);  
 }  
  
 if (pse = getservbyname(service, transport))  
 sin.sin\_port = pse->s\_port;  
 else if ((sin.sin\_port = htons((u\_short) atoi(service))) == 0)  
 errexit("can't get \"%s\" service entry\n", service);  
/\* Map host name to IP address, allowing for dotted decimal \*/  
 if (phe = gethostbyname(host))  
 memcpy(&sin.sin\_addr, phe->h\_addr, phe->h\_length);  
  
 else if ((sin.sin\_addr.s\_addr = inet\_addr(host)) == INADDR\_NONE)  
 errexit("can't get \"%s\" host entry\n", host);  
/\* Map transport protocol name to protocol number \*/  
 if ((ppe = getprotobyname(transport)) == 0)  
 errexit("can't get \"%s\" protocol entry\n", transport);  
/\* Use protocol to choose a socket type \*/  
 if (strcmp(transport, "udp") == 0)  
 type = SOCK\_DGRAM;  
 else  
 type = SOCK\_STREAM;  
/\* Allocate a socket \*/  
 s = socket(PF\_INET, type, ppe->p\_proto);  
 if (s < 0)  
 errexit("can't create socket: %s\n", strerror(errno));  
/\* Connect the socket \*/  
 if (connect(s, (struct sockaddr \*) &sin, sizeof(sin)) < 0)  
 errexit("can't connect to %s.%s: %s\n", host, service,  
 strerror(errno));  
  
 (void) write(s, MSG, strlen(MSG));  
  
 /\* Read the time \*/  
 n = read(s, (char \*) &now, sizeof(now));  
 if (n < 0)  
 errexit("read failed: %s\n", strerror(errno));  
 now = ntohl((u\_long) now); /\* put in host byte order \*/  
 now -= UNIXEPOCH; /\* convert UCT to UNIX epoch \*/  
 printf("%s", ctime(&now));  
 exit(0);  
}

---------------------------------------------------------------------------------------------------

**END OF QUESTION 1**

---------------------------------------------------------------------------------------------------

---------------------------------------------------------------------------------------------------

**QUESTION 2**

---------------------------------------------------------------------------------------------------

**Readme:**

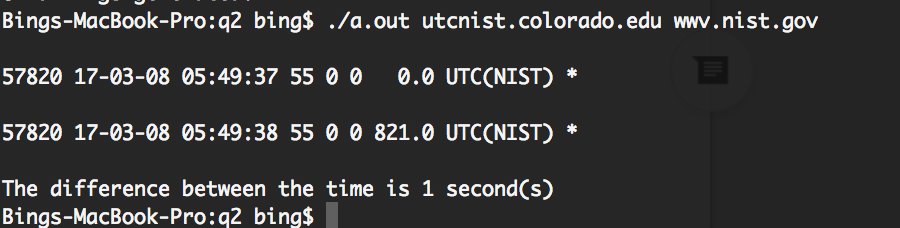
To compile:

1. make

OR

2. gcc TCPDayTime.c

To run(test example): ./a.out utcnist.colorado.edu wwv.nist.gov

The result:

To clean the files:

1. make clean

OR

2. rm -f a.out

**Makefile:**

all:

gcc TCPDayTime.c

clean:

rm -f a.out

**Code:**

/\* TCPdaytime.c - TCPdaytime, main \*/  
#include <unistd.h>  
#include <stdlib.h>  
#include <string.h>  
#include <stdio.h>  
#include <stdarg.h>  
#include <errno.h>  
#include <sys/socket.h>  
#include <netinet/in.h>  
#include <arpa/inet.h>  
#include <time.h>  
#include <netdb.h>  
#include <ctype.h>  
  
#define LINELEN 128  
#ifndef INADDR\_NONE  
#define INADDR\_NONE 0xffffffff  
#endif /\* INADDR\_NONE \*/  
  
//extern int errno;  
int TCPdaytime(const char \*host, const char \*service);  
  
int errexit(const char \*format, ...);  
  
int connectTCP(const char \*host, const char \*service);  
  
typedef unsigned short u\_short;  
  
/\*------------------------------------------------------------------------  
 \* main - TCP client for DAYTIME service  
 \*------------------------------------------------------------------------  
 \*/  
int  
main(int argc, char \*argv[]) {  
 char \*host1 = "localhost"; /\* host to use if none supplied \*/  
 char \*host2 = "localhost"; /\* default service port \*/  
 switch (argc) {  
 case 1:  
 host1 = "localhost";  
 host2 = "localhost";  
 break;  
 case 3:  
 host1 = argv[1];  
 host2 = argv[2];  
/\* FALL THROUGH \*/  
 case 2:  
 host1 = argv[1];  
 host2 = argv[2];  
 break;  
 default:  
 fprintf(stderr, "usage: TCPdaytime [host [port]]\n");  
 exit(1);  
 }  
 TCPdaytime(host1, host2);  
  
 exit(0);  
}  
  
/\*------------------------------------------------------------------------  
 \* TCPdaytime - invoke Daytime on specified host and print results  
 \*------------------------------------------------------------------------  
 \*/  
int TCPdaytime(const char \*host1, const char \*host2) {  
 char buf[LINELEN + 1];  
 char buf2[LINELEN + 1];  
 int s1, s2; /\* socket, read count \*/  
 ssize\_t n;  
 char \*time1, \*time2 = NULL;  
 int len = 0;  
  
 s1 = connectTCP(host1, "daytime");  
 while ((n = read(s1, buf, LINELEN)) > 0) {  
 time1 = realloc(time1, len + n);  
 memcpy(time1 + len, buf, n);  
 len += n;  
  
 } /\* to get daytime from host1 \*/  
  
 time1[len] = '\0';  
 printf("%s", time1);  
  
 len = 0;  
 n = 0;  
 s2 = connectTCP(host2, "daytime");  
  
 while ((n = read(s2, buf, LINELEN)) > 0) {  
 time2 = realloc(time2, len + n);  
 memcpy(time2 + len, buf, n);  
 len += n;  
  
 }/\* to get daytime from host2 \*/  
  
 time2[len] = '\0';  
 printf("%s", time2);  
  
  
 // differences between host1, host2  
 struct tm result, result2;  
 char formatedTime1[22] = {0};  
 char formatedTime2[22] = {0};  
  
 strcat(strncpy(formatedTime1, (time1 + 7), 17), " 0000");  
 strcat(strncpy(formatedTime2, (time2 + 7), 17), " 0000");  
  
 memset(time1, 0, sizeof(time1));  
 memset(time2, 0, sizeof(time2));  
 free(time1);  
 free(time2);  
  
 strptime(formatedTime1, "%y-%m-%d %H:%M:%S %z", &result);  
 strptime(formatedTime2, "%y-%m-%d %H:%M:%S %z", &result2);  
 int diff = abs(timegm(&result) - timegm(&result2));  
  
// printf("%lu, %lu", mktime(&result), mktime(&result2));  
// printf("%d, %d", result.tm\_hour, result2.tm\_hour);  
 printf("\nThe difference between the time is %d second(s)\n", diff);  
}  
  
int  
errexit(const char \*format, ...) {  
 va\_list args;  
 va\_start(args, format);  
 vfprintf(stderr, format, args);  
 va\_end(args);  
 exit(1);  
}  
  
int connectsock(const char \*host, const char \*service,  
 const char \*transport);  
  
/\*------------------------------------------------------------------------  
 \* connectTCP - connect to a specified TCP service on a specified host  
 \*------------------------------------------------------------------------  
 \*/  
int  
connectTCP(const char \*host, const char \*service)  
/\*  
 \* Arguments:  
 \* host - name of host to which connection is desired  
 \* service - service associated with the desired port  
 \*/  
{  
 return connectsock(host, service, "tcp");  
}  
  
/\*------------------------------------------------------------------------  
 \* connectsock - allocate & connect a socket using TCP or UDP  
 \*------------------------------------------------------------------------  
 \*/  
int  
connectsock(const char \*host, const char \*service, const char \*transport)  
/\*  
 \* Arguments:  
 \* host - name of host to which connection is desired  
 \* service - service associated with the desired port  
 \* transport - name of transport protocol to use ("tcp" or "udp")  
 \*/  
{  
 struct hostent \*phe; /\* pointer to host information entry \*/  
 struct servent \*pse; /\* pointer to service information entry \*/  
 struct protoent \*ppe; /\* pointer to protocol information entry\*/  
 struct sockaddr\_in sin; /\* an Internet endpoint address \*/  
 int s, type; /\* socket descriptor and socket type \*/  
 memset(&sin, 0, sizeof(sin));  
 sin.sin\_family = AF\_INET;  
 /\* Map service name to port number \*/  
 if (pse = getservbyname(service, transport))  
 sin.sin\_port = pse->s\_port;  
 else if ((sin.sin\_port = htons((u\_short) atoi(service))) == 0)  
 errexit("can't get \"%s\" service entry\n", service);  
 /\* Map host name to IP address, allowing for dotted decimal \*/  
 if (phe = gethostbyname(host))  
 memcpy(&sin.sin\_addr, phe->h\_addr, phe->h\_length);  
 else if ((sin.sin\_addr.s\_addr = inet\_addr(host)) == INADDR\_NONE)  
 errexit("can't get \"%s\" host entry\n", host);  
 /\* Map transport protocol name to protocol number \*/  
 if ((ppe = getprotobyname(transport)) == 0)  
 errexit("can't get \"%s\" protocol entry\n", transport);  
 /\* Use protocol to choose a socket type \*/  
 if (strcmp(transport, "udp") == 0)  
 type = SOCK\_DGRAM;  
 else  
 type = SOCK\_STREAM;  
 /\* Allocate a socket \*/  
 s = socket(PF\_INET, type, ppe->p\_proto);  
 if (s < 0)  
 errexit("can't create socket: %s\n", strerror(errno));  
 /\* Connect the socket \*/  
 if (connect(s, (struct sockaddr \*) &sin, sizeof(sin)) < 0)  
 errexit("can't connect to %s.%s: %s\n", host, service,  
 strerror(errno));  
 return s;  
}

---------------------------------------------------------------------------------------------------

**END OF QUESTION 2**

---------------------------------------------------------------------------------------------------

---------------------------------------------------------------------------------------------------

**QUESTION 3**

---------------------------------------------------------------------------------------------------

**Question 3**

TCP Client-Server for getting count of characters,

**Readme:**

To compile:

1. make

OR

2. gcc -o server EchoServer.c

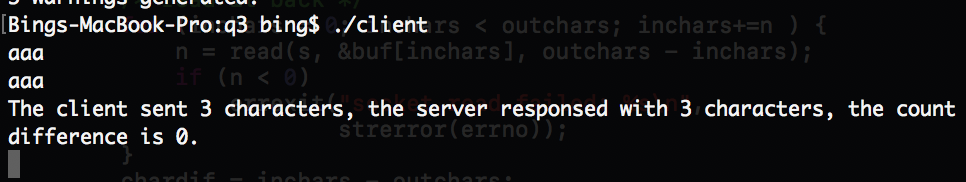
gcc -o client TCPecho.c

To run:

1. Run server: ./server

(May need administrator/root access)

1. Open another terminal
2. Run client: ./client
3. Test example: type “aaa”

The result:

To clean the files:

1. make clean

OR

2. rm -f server client

**Makefile**:

all: server client

server:

gcc -o server EchoServer.c

client:

gcc -o client TCPecho.c

clean:

rm -f server client

**Server code**:

//  
// EchoServer.c  
//   
//  
// Created by Bing Shi on 3/2/17.  
//  
//  
/\* EchoServer.c - main, TCPechod \*/  
#include <sys/types.h>  
#include <sys/signal.h>  
#include <sys/socket.h>  
#include <sys/time.h>  
#include <sys/resource.h>  
#include <sys/wait.h>  
#include <sys/errno.h>  
#include <netinet/in.h>  
#include <unistd.h>  
#include <stdlib.h>  
#include <stdio.h>  
#include <string.h>  
#include <errno.h>  
#include <netdb.h>  
#include <stdarg.h>  
#include <arpa/inet.h>  
#define QLEN 5 /\* maximum connection queue length \*/  
#define BUFSIZE 4096  
u\_short portbase = 0; /\* port base, for non-root servers \*/  
void reaper(int);  
int TCPechod(int fd);  
int errexit(const char \*format, ...);  
int passiveTCP(const char \*service, int qlen);  
/\*------------------------------------------------------------------------  
 \* main - Concurrent TCP server for ECHO service  
 \*------------------------------------------------------------------------  
 \*/  
int  
main(int argc, char \*argv[])  
{  
 char \*service = "echo"; /\* service name or port number \*/  
 struct sockaddr\_in fsin; /\* the address of a client \*/  
 int alen; /\* length of client's address \*/  
 int msock; /\* master server socket \*/  
 int ssock; /\* slave server socket \*/  
 switch (argc) {  
 case 1:  
 break;  
 case 2:  
 service = argv[1];  
 break;  
 default:  
 errexit("usage: TCPechod [port]\n");  
 }  
 msock = passiveTCP(service, QLEN);  
 (void) signal(SIGCHLD, reaper);  
 while (1) {  
 alen = sizeof(fsin);  
 ssock = accept(msock, (struct sockaddr \*)&fsin, &alen);  
 if (ssock < 0) {  
 if (errno == EINTR)  
 continue;  
 errexit("accept: %s\n", strerror(errno));  
 }  
 switch (fork()) {  
 case 0: /\* child \*/  
 (void) close(msock);  
 exit(TCPechod(ssock));  
 default: /\* parent \*/  
 (void) close(ssock);  
 break;  
 case -1:  
 errexit("fork: %s\n", strerror(errno));  
 }  
 }  
}

/\*------------------------------------------------------------------------  
 \* TCPechod - echo data until end of file  
 \*------------------------------------------------------------------------  
 \*/  
int TCPechod(int fd)  
{  
 char buf[BUFSIZ];  
 int cc;  
 while (cc = read(fd, buf, sizeof buf)) {  
 if (cc < 0)  
 errexit("echo read: %s\n", strerror(errno));  
 if (write(fd, buf, cc) < 0)  
 errexit("echo write: %s\n", strerror(errno));  
 }  
 return 0;  
}  
/\*------------------------------------------------------------------------  
 \* reaper - clean up zombie children  
 \*------------------------------------------------------------------------  
 \*/  
/\*ARGSUSED\*/  
void  
reaper(int sig)  
{  
 int status;  
 while (wait3(&status, WNOHANG, (struct rusage \*)0) >= 0)  
 /\* empty \*/;  
}  
//int passivesock(const char \*service, const char \*transport,  
// int qlen);  
/\*------------------------------------------------------------------------  
 \* passiveTCP - create a passive socket for use in a TCP server  
 \*------------------------------------------------------------------------  
 \*/  
int  
passiveTCP(const char \*service, int qlen)  
/\*  
 \* Arguments:  
 \* service - service associated with the desired port  
 \* qlen - maximum server request queue length  
 \*/  
{  
 return passivesock(service, "tcp", qlen);  
}  
int  
errexit(const char \*format, ...)  
{  
 va\_list args;  
 va\_start(args, format);  
 vfprintf(stderr, format, args);  
 va\_end(args);  
 exit(1);  
}  
/\*------------------------------------------------------------------------  
 \* passivesock - allocate & bind a server socket using TCP or UDP  
 \*------------------------------------------------------------------------  
 \*/  
int  
passivesock(const char \*service, const char \*transport, int qlen)  
/\*  
 \* Arguments:  
 \* service - service associated with the desired port  
 \* transport - transport protocol to use ("tcp" or "udp")  
 \* qlen - maximum server request queue length  
 \*/  
{  
 struct servent \*pse; /\* pointer to service information entry \*/  
 struct protoent \*ppe; /\* pointer to protocol information entry\*/  
 struct sockaddr\_in sin; /\* an Internet endpoint address \*/  
 int s, type; /\* socket descriptor and socket type \*/  
 memset(&sin, 0, sizeof(sin));  
 sin.sin\_family = AF\_INET;  
 sin.sin\_addr.s\_addr = INADDR\_ANY;  
 /\* Map service name to port number \*/  
 if ( pse = getservbyname(service, transport) )  
 sin.sin\_port = htons(ntohs((u\_short)pse->s\_port)  
 + portbase);  
 else if ( (sin.sin\_port = htons((u\_short)atoi(service))) == 0 )  
 errexit("can't get \"%s\" service entry\n", service);  
 /\* Map protocol name to protocol number \*/  
 if ( (ppe = getprotobyname(transport)) == 0)  
 errexit("can't get \"%s\" protocol entry\n", transport);  
 /\* Use protocol to choose a socket type \*/  
 if (strcmp(transport, "udp") == 0)  
 type = SOCK\_DGRAM;  
 else  
 type = SOCK\_STREAM;  
 /\* Allocate a socket \*/  
 s = socket(PF\_INET, type, ppe->p\_proto);  
 if (s < 0)  
 errexit("can't create socket: %s\n", strerror(errno));  
 /\* Bind the socket \*/  
 if (bind(s, (struct sockaddr \*)&sin, sizeof(sin)) < 0)  
 errexit("can't bind to %s port: %s\n", service,  
 strerror(errno));  
 if (type == SOCK\_STREAM && listen(s, qlen) < 0)  
 errexit("can't listen on %s port: %s\n", service,  
 strerror(errno));  
 return s;  
}// end of server code

**Client code:**

//

// TCPecho.c

//

//

// Created by Bing Shi on 3/3/17.

//

//

/\* TCPecho.c - main, TCPecho \*/

#include <unistd.h>

#include <stdlib.h>

#include <string.h>

#include <stdio.h>

#include <stdarg.h>

#include <errno.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <netdb.h>

#include <ctype.h>

#define LINELEN 128

#define \_\_USE\_BSD 1

#ifndef INADDR\_NONE

#define INADDR\_NONE 0xffffffff

#endif /\* INADDR\_NONE \*/

int TCPecho(const char \*host, const char \*service);

int errexit(const char \*format, ...);

int connectTCP(const char \*host, const char \*service);

/\*------------------------------------------------------------------------

\* main - TCP client for ECHO service

\*------------------------------------------------------------------------

\*/

int

main(int argc, char \*argv[])

{

char \*host = "localhost"; /\* host to use if none supplied \*/

char \*service = "echo"; /\* default service name \*/

switch (argc) {

case 1:

host = "localhost";

break;

case 3:

service = argv[2];

/\* FALL THROUGH \*/

case 2:

host = argv[1];

break;

default:

fprintf(stderr, "usage: TCPecho [host [port]]\n");

exit(1);

}

TCPecho(host, service);

exit(0);

}

/\*------------------------------------------------------------------------

\* TCPecho - send input to ECHO service on specified host and print reply

\*------------------------------------------------------------------------

\*/

int

TCPecho(const char \*host, const char \*service)

{

char buf[LINELEN+1]; /\* buffer for one line of text \*/

int s, n; /\* socket descriptor, read count\*/

int outchars, inchars, chardif; /\* characters sent and received \*/

s = connectTCP(host, service);

while (fgets(buf, sizeof(buf), stdin)) {

buf[LINELEN] = '\0'; /\* insure line null-terminated \*/

outchars = strlen(buf);

(void) write(s, buf, outchars);

/\* read it back \*/

for (inchars = 0; inchars < outchars; inchars+=n ) {

n = read(s, &buf[inchars], outchars - inchars);

if (n < 0)

errexit("socket read failed: %s\n",

strerror(errno));

}

chardif = inchars - outchars;

fputs(buf, stdout);

char buf1[] = "The client sent %d characters, the server responsed with %d characters, the count difference is %d.\n";

printf(buf1, outchars -1 , inchars -1, chardif);

}

}

int

errexit(const char \*format, ...)

{

va\_list args;

va\_start(args, format);

vfprintf(stderr, format, args);

va\_end(args);

exit(1);

}

int connectsock(const char \*host, const char \*service,

const char \*transport);

/\*------------------------------------------------------------------------

\* connectTCP - connect to a specified TCP service on a specified host

\*------------------------------------------------------------------------

\*/

int

connectTCP(const char \*host, const char \*service )

/\*

\* Arguments:

\* host - name of host to which connection is desired

\* service - service associated with the desired port

\*/

{

return connectsock( host, service, "tcp");

}

/\*------------------------------------------------------------------------

\* connectsock - allocate & connect a socket using TCP or UDP

\*------------------------------------------------------------------------

\*/

int

connectsock(const char \*host, const char \*service, const char \*transport )

/\*

\* Arguments:

\* host - name of host to which connection is desired

\* service - service associated with the desired port

\* transport - name of transport protocol to use ("tcp" or "udp")

\*/

{

struct hostent \*phe; /\* pointer to host information entry \*/

struct servent \*pse; /\* pointer to service information entry \*/

struct protoent \*ppe; /\* pointer to protocol information entry\*/

struct sockaddr\_in sin; /\* an Internet endpoint address \*/

int s, type; /\* socket descriptor and socket type \*/

memset(&sin, 0, sizeof(sin));

sin.sin\_family = AF\_INET;

/\* Map service name to port number \*/

if ( pse = getservbyname(service, transport) )

sin.sin\_port = pse->s\_port;

else if ( (sin.sin\_port = htons((u\_short)atoi(service))) == 0 )

errexit("can't get \"%s\" service entry\n", service);

/\* Map host name to IP address, allowing for dotted decimal \*/

if ( phe = gethostbyname(host) )

memcpy(&sin.sin\_addr, phe->h\_addr, phe->h\_length);

else if ( (sin.sin\_addr.s\_addr = inet\_addr(host)) == INADDR\_NONE )

errexit("can't get \"%s\" host entry\n", host);

/\* Map transport protocol name to protocol number \*/

if ( (ppe = getprotobyname(transport)) == 0)

errexit("can't get \"%s\" protocol entry\n", transport);

/\* Use protocol to choose a socket type \*/

if (strcmp(transport, "udp") == 0)

type = SOCK\_DGRAM;

else

type = SOCK\_STREAM;

/\* Allocate a socket \*/

s = socket(PF\_INET, type, ppe->p\_proto);

if (s < 0)

errexit("can't create socket: %s\n", strerror(errno));

/\* Connect the socket \*/

if (connect(s, (struct sockaddr \*)&sin, sizeof(sin)) < 0)

errexit("can't connect to %s.%s: %s\n", host, service,

strerror(errno));

return s;

}

---------------------------------------------------------------------------------------------------

**END OF QUESTION 3**

---------------------------------------------------------------------------------------------------