Message Packetization

(read()) and write() just takes char* or void* as second argument. Means, in order to send/receive your SSN:

```c
int SSN;
write(s, (char *) &SSN, sizeof (int));
read(s, (char *) &SSN, sizeof (int));
```

What if your message consists of many fields of different types such as: <first name (string), last name (string), grade (double), ID (int)>. Two ways to package your message (details later):

- As a string using sprintf
- As a struct

What if you have more than type of message such as addCourse, dropCourse, checkTerm, .. etc. In order to classify incoming messages we need to add msgType (or reqCode) in the prefix of the message before sending. The receiver reads this field first to know how to read the remaining part and how to decode the received message.

For example,

- To add a course <ADD, FName, Lname, Id, Course>
- To drop a course <DROP, FName, Lname, Id, Course>
- To list courses <LIST, FName, Lname, Id, Term>

The first field determines the structure of the message.

In TCP, the receiver always needs to know the size of the incoming messages in order to read and process messages independently. But TCP is stream oriented and does not preserve the message boundaries; it sends out bytes according to the network flow/congestion control (segmentation):

- Write(s, buf, 100) but TCP might send in the network:
  - 50 50 20 means need two reads, OR
  - 30 20 30 means need three reads

Also (repacketization):

- Write(s, buf, 100); write(s, buf, 100) 200 (one read)
Message Packetization

- Because of this segmentation and re-packetization, receiver does not know how many read() it needs in order to read the full message.
- Therefore, in TCP, you should ALWAYS read in a loop until the entire length is read or the remote party closes the connection. This also implies that the message length should be known by the receiver before reading the message (either hardcoded or encoded in the message itself)
- In UDP, the receiver’s read() (or receive()) reads (if any) a complete message size. So you do not need to worry about this issue in UDP!

Two ways to package a message:

1. As a combined string using sprintf():
   - Need to append the message length field and separators between strings in the message.
2. As a struct (more professional): the length is not needed because you are sending the entire struct size any way and the receiver knows the size from the shared header files. However, msgType field is needed. For example,

   ```c
   struct MSG {
   int    msgType;
   char Lname[32];
   char Fname[32];
   
   msg.msgType=1; Strcpy(msg.Lname,"Al-Shaer");
   
   struct MSG {
   int    msgType;
   char Lname[32];
   char Fname[32];
   
   printf("%d
", sizeof (msg));
   write(s, (void*) &msg, sizeof (msg));
   ```

Reading the entire message byte-by-byte is not a good idea.

Using sprintf, the sender will do:

```c
int ID;
double Grade;
char FName[32],Lname[32], buf[1024];
Length=4+4+6+strlen(Lname)+strlen(Fname);

sprintf(buf, "%d%d%s-%s-%6.2f", 
   length,ID,Fname,Lname,Grade);
write(s, buf, length+2);
```

The TCP receiver should do:

```c
size=sizeof(int); /* to read the length */
char len[4], msg[1024];

len=bufptr;
```
Packetization Using `sprintf`

```c
while((n=read(s, bufptr, size)) >0)
{ size-=n;  bufptr+=n;  
ilen=atoi(len);
bufptr=msg;
while((n=read(s, bufptr, ilen)) > 0)
{ ilen-=n;  bufptr+=n;  
scanf(msg,¨%4d%4d%-%%-%-%%,
ID,Fname,Lname,Grade);
}
```

But the UDP receiver should do:

```c
read(s, msg, maxLen);
sccanf(msg,¨%4d%4d%-%%-%-%%, Length,
ID,Fname,Lname,Grade);
```

Packetization Using `struct`

```c
#else  enter the same struct format and does:

```c
int msgType;
bufptr=(char*)&msgType; size=4;
while((n=read(s, bufptr, size)) >0)
{ size-=n;  bufptr+=n;  
if (msgType == ADDMSG)
{  
addptr = (addrCourse)+4; size=sizeof (addCourse)-4
while((n=read(s, (void*) addptr, size)) >0)
{ size-=n;  addptr+=n;  
}
else if (msgtype == LISTMSG)
{  /* same as above */  }  
```
Packetization Using struct
(for one message format)

In UDP Receiver, if you expect ONE format (struct) for more than one message (e.g., add and drop shares the same struct):

```c
regCourse mymsg;
read(s, (char*) &mymsg, MAXMSGLEN); /* One read!*/
if (mymsg.msgType == ADDMSG)
    { printf("%d, %s, %s, %d %d %s
", mymsg.ID,
    mymsg.Fname, mymsg.Lname, mymsg.numCourses,
    mymsg.courseList.section, mymsg.courseList.name);
    AddCourse(&mymsg);
    }
else if (msg.msgType == DROPMSG)
    { /* similar stuff */
    DropCourse(&mymsg);
    }
```

Packetization Using struct
(for different message formats)

UDP Receiver if every message has a different format (struct):

```c
read(s, msg, MAXMSGLEN); /* MUST read at once!! */
strncpy((char*)&msgtype, msg, 4);
if (msgtype == ADDMSG)
    { regCourse mymsg;
        strcpy(&mymsg, msg);
        printf("%d, %s, %s, %d %d %s
", mymsg.ID,
        mymsg.Fname, mymsg.Lname, mymsg.numCourses,
        mymsg.courseList.section, mymsg.courseList.name);
    }
else if (msgtype == LISTMSG)
    { /* similar stuff */
    }
```

Packetization Using struct with
Unbounded List

To Send the following message,

```c
struct REPLY { int count; /* numOfCourses */ int *Course; /* List of courses numbers */
} Reply;
```
you do the following:

```c
int Courses[10], LisySize, ReplySize, count;
char *pptr;
LisySize = sizeof(int)* count;
ReplySize = LisySize + sizeof(int);
pptr = (char*) malloc(ReplySize);
memcpy(pptr, &count, sizeof(int)); /* to put the count*/
memcpy(pptr+4, Courses, LisySize); /* to put the avail courses*/
write(s, (void *) pptr, ReplySize);
```