CO650 Advanced Programming - TCP

C++ Transmitting Data (TCP)

TCP requires connection to be established - it will only send data through a socket which connects both client and server.

Sending Data

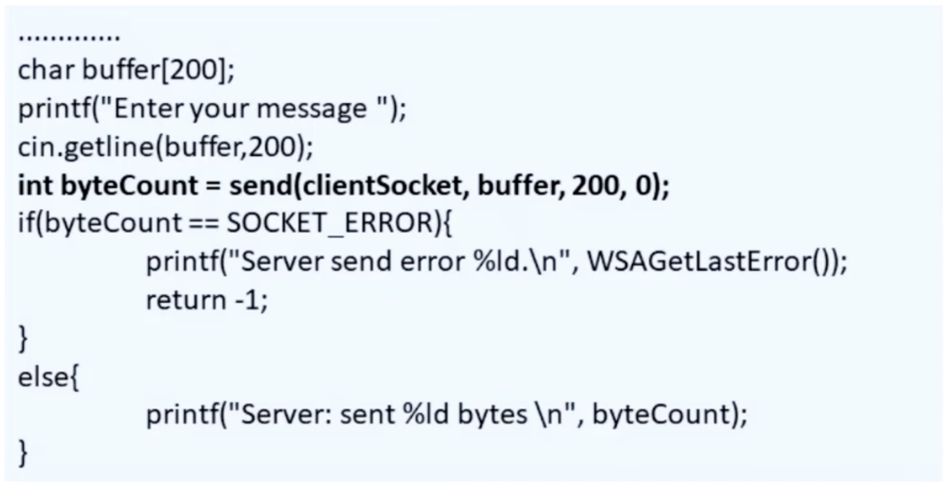
- The send function sends data on a connected socket

```
int send(SOCKET s, const char *buf, int len, int flags);
```

s	The descriptor that identifies a connected socket.
buf	A pointer to the buffer to the data to be transmitted.
len	The length, in bytes, of the buffer pointed to by the buf parameter
flags	Optional set of flags that influences the behavior of this function (No routing, etc.)

If no error occurs, send returns the number of bytes sent. Otherwise SOCKET_ERROR is returned.

Send Example



```
.............
char buffer[200];
printf("Enter your message ");
cin.getline(buffer,200);
int byteCount = send(clientSocket, buffer, 200, 0);
if(byteCount == SOCKET_ERROR){
            printf("Server send error %ld.\n", WSAGetLastError());
            return -1;
}
else{
            printf("Server: sent %ld bytes \n", byteCount);
}
```

User Input

- Include **<iostream>** and **using namespace std**
- getline - this function is part of the cin library. User Input is taken through this method instead of *cin* command because using *cin* would terminate at white space and therefore typically used to read a single word at a time. In this case getline would be more appropriate because ideally we would want more than one word in case of writing sentences.
- Pass a valid char array and length of the array to getLine as arguments.

```
char buffer[200] = “”;
cout << “Enter your message”;
cin.getline(buffer,200);
cout “You typed” << buffer << endl;
```

Receiving Data

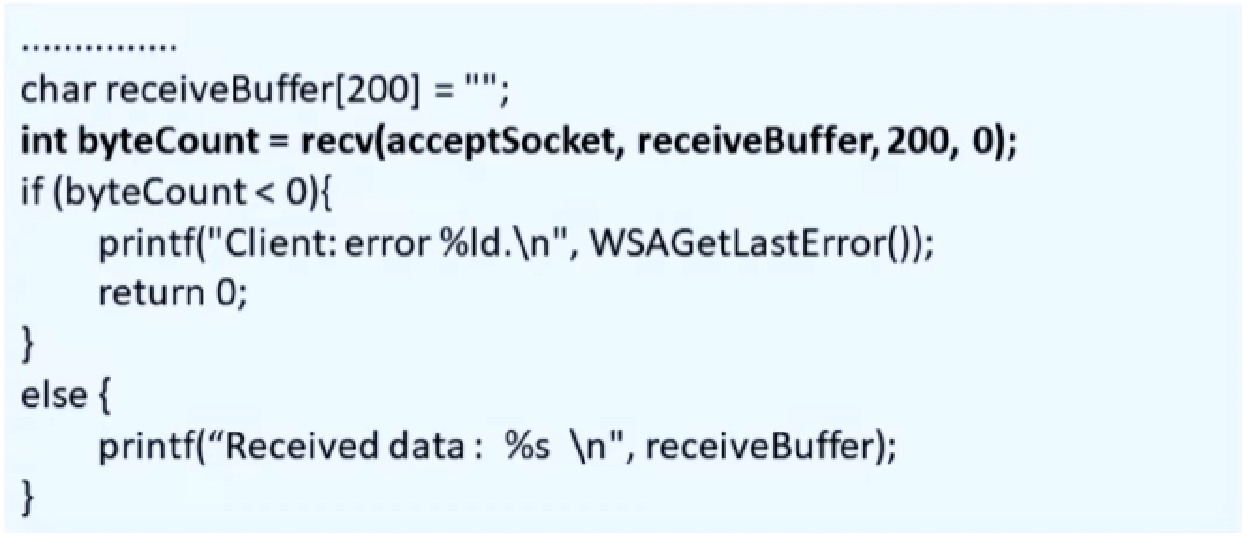
- The **recv** function receives data from a connected socket

```
int recv(SOCKET s, char *buf, int len, int flags);
```

s	The descriptor that identifies a connected socket
buf	A pointer to the buffer to receive to receive the incoming data
len	Then length, in bytes, of the buffer pointed to by the buf parameter
flags	Optional set of flags that influences the behavior of this function

If no error occurs, **recv** returns the number of bytes received. If the connection has been gracefully closed, the return value is zero. Otherwise SOCKET_ERROR is returned.

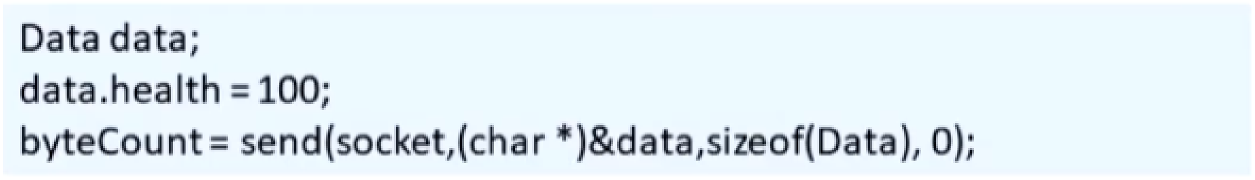
Receive Example



```
................
char receiveBuffer[200] = "";
int byteCount = recv(acceptSocket, receiveBuffer, 200, 0);
if (byteCount < 0){
      printf("Client: error %ld.\n", WSAGetLastError());
      return 0;
}
else {
      printf("Received data :  %s  \n", receiveBuffer);
}
```

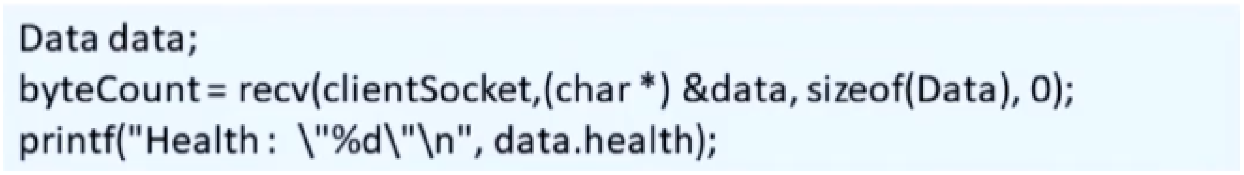
Transmitting Objects

Sending Object Example



```
Data data;
data.health = 100;
byteCount = send(socket,(char *)&data,sizeof(Data), 0);
```

Receiving Object Example



```
Data data;
byteCount = recv(clientSocket,(char *) &data, sizeof(Data), 0);
printf("Health :  \"%d\"\n", data.health);
```

Assigning Values to Char Array

When defining the character array we can initialize its value

```
char sendBuffer[200] = “Message received by server”;
```

However, we can’t use the assignment operator after it has been defined. Instead we must use the strcpy function or if depreciated strcpy_s.

```
char buffer[200];
strcpy_s(buffer,”hello world”);
```

Use the strlen function to return the number of characters in the array.

```
strlen(sendBuffer);
```

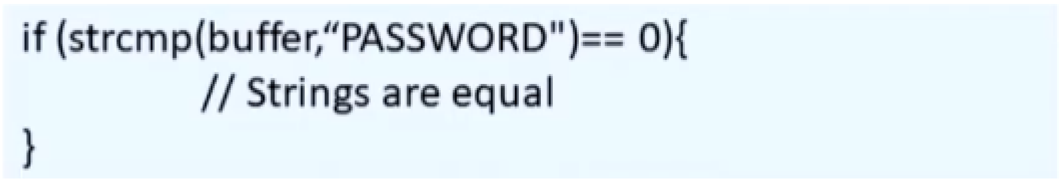
Comparing Character Arrays

- The strcmp() function compares two strings

```
int strcmp(const char* string1, const  char* string2);
```

- Takes two strings as arguments (pointers to character arrays)
- Returns 0 if they are equal
- Returns < 0 if string1 is less than string2
- Returns > 0 if string1 is greater than string2

strcmp() Example



```
if (strcmp(buffer,“PASSWORD")== 0){
            // Strings are equal
}
```