```c
#define BUF_SIZE 200
#define PORT 8080
#define SERVER_IP "192.0.2.1"

int main(void) {
    int sock = socket(AF_INET, SOCK_STREAM, 0);
    if (sock < 0) { perror("socket"); exit(1); }

    struct sockaddr_in addr;
    memset(&addr, 0, sizeof(addr));
    addr.sin_family = AF_INET;
    addr.sin_port   = htons(PORT);
    inet_pton(AF_INET, SERVER_IP, &addr.sin_addr);

    if (connect(sock, (struct sockaddr *)&addr, sizeof(addr)) < 0) {
        perror("connect"); exit(1);
    }

    char *msg  = "the quick brown fox";   /* strlen is 19 */
    int msg_len = strlen(msg);

    int sent = send(sock, msg, msg_len, 0);                 /* line A */

    char buf[BUF_SIZE];
    int n = recv(sock, buf, BUF_SIZE, 0);                   /* line B */
    buf[n] = '\0';                                          /* line C */
    printf("echoed: %s\n", buf);

    close(sock);
    return 0;
```

## API reference

```c
int  socket(int domain, int type, int protocol);
int  connect(int sockfd, const struct sockaddr *addr,
             socklen_t addrlen);
ssize_t send(int sockfd, const void *msg, size_t msgLength, int flags);
ssize_t recv(int sockfd, void *rcvBuffer, size_t bufferLength, int flags);
int close(int sockfd);
```

`send()` and `recv()` return the number of bytes copied, or -1 on failure. `recv()` returns 0 when the peer has closed the connection.

# [CS 43] Sockets Worksheet

You have a server running that reads bytes from a client, echoes the same bytes back, and then closes the connection.

## Part 1: Read the provided client code

This is not a complete program (#include headers are omitted). This program has at least one bug. Answer the following questions about the code.

**1. Descriptor table.** The left column is what the kernel is storing for this socket the instant `socket()` returns. Fill in the right column for what the kernel is storing after `connect()` returns.

| | **after `socket()` returns** | **after `connect()` returns** |
|---|---|---|
| value of `sock` | an int (ex: 7) | |
| family | `AF_INET` | |
| type | `SOCK_STREAM` | |
| local IP address | not set | |
| local port | not set | |
| remote IP address | not set | |
| remote port | not set | |
| send buffer | empty | |
| receive buffer | empty | |

**2.** Did `connect()` give you a new descriptor?

**3. Line A returns 19.** Which of these is guaranteed? Circle all that apply.

A. The kernel has accepted 19 bytes into the send buffer

B. All 19 bytes have left this machine

C. All 19 bytes have arrived at the server

D. The server application has read all 19 bytes

**4. Line B: three scenarios.** For each, give the value of `n` and the exact string this program prints.

| State of the receive socket buffer when line B runs | n | Printed |
|---|---|---|
| (a) empty, the server's 19 bytes arrive a couple of milliseconds later | | |
| (b) holds the first 8 bytes; the other 11 arrive a couple of milliseconds later | | |
| (c) holds all 19 bytes | | |

**5. The bug.** Scenario (b) is not unusual. Explain in one sentence why the program is wrong.

**6. Lines A, B and C.** Besides the bug you just described, there are three more problems in this code. Find at least two. (Hint: what are all the values `recv()` and `send()` can return, and how big is `buf`?

# Part 2: Fix it

## 7. Receive loop

The client knows it sent 19 bytes, so it knows to expect 19 back. Fill in the blanks so the client keeps receiving until it has all of them.

```
01  int total = 0;
02  int n;
03  while (total < msg_len) {
04      n = recv(sock, _____________, _____________, 0);
05      if (n < 0)  { perror("recv"); exit(1); }
06      if (n == 0) { fprintf(stderr, "server closed early\n"); break; }
07      total += n;
08  }
09  buf[total] = '\0';
```

Second blank: be careful. `buf` is `BUF_SIZE` bytes and you are going to write a terminator at `buf[total]`. What is the largest number of bytes it is safe to ask for on a call where `total` bytes are already in the buffer?

## 8. Send loop

`send()` can also copy fewer bytes than you asked for. Write the loop.

```
ssize_t send_all(int sock, const char *data, size_t len) {
    size_t total = 0;
    while (____________________) {
        ssize_t n = send(sock, ______________, ______________, 0);
        if (n < 0) { perror("send"); return -1; }
        total += n;
    }
    return total;
```

**9.** In question 7 we wrote `if (n == 0) break;` and called it an error. Why is a 0 return an error for the echo client but not for a client that is fetching a web page?

## 10. Extend: the web client (Do this on your own, after class)

Now the length is unknown. You are fetching an object over HTTP, and the server closes the connection when it is finished sending. You may assume the files you'll be retrieving are no larger than one megabyte, and that `buf` has been declared large enough to hold one.

Rewrite the receive loop for this case. State your loop condition and your exit condition explicitly before you write any code.

```
size_t total = 0;
/* loop condition: ____________________________________________ */
/* exit condition: ____________________________________________ */
```