The following excerpt of code is taken from an exercise I performed for a computer networks course at UBC. The 7 routines below makeup the main functionality of an alternating-bit protocol, and are called on by a data-link layer simulator program.

```
#define BUFSIZE 64

struct Sender {
    int base;
    int nextseq;
    int window_size;
    float estimated_rtt;
    int buffer_next;
    struct pkt packet_buffer[BUFSIZE];
} A;

struct Receiver {
    int expect_seq;
    struct pkt packet_to_send;
} B;

int get_checksum(struct pkt *packet) {
    int checksum = 0;
    checksum += packet->seqnum;
    checksum += packet->acknum;
    for (int i = 0; i < 20; ++i)
        checksum += packet->payload[i];
    return checksum;
}

void send_window(void) {
    while (A.nextseq < A.buffer_next && A.nextseq < A.base + A.window_size)
{
        struct pkt *packet = &A.packet_buffer[A.nextseq % BUFSIZE];
        printf("  send_window: send packet (seq=%d): %s\n", packet->seqnum,
packet->payload);
        tolayer3(0, *packet);
        if (A.base == A.nextseq)
            starttimer(0, A.estimated_rtt);
        ++A.nextseq;
    }
}

/* called from layer 5, passed the data to be sent to other side */
void A_output(struct msg message) {
    if (A.buffer_next - A.base >= BUFSIZE) {
        printf("  A_output: buffer full. drop the message: %s\n",
message.data);
        return;
    }
```

```
    printf("  A_output: bufferred packet (seq=%d): %s\n", A.buffer_next,
message.data);
    struct pkt *packet = &A.packet_buffer[A.buffer_next % BUFSIZE];
    packet->seqnum = A.buffer_next;
    memmove(packet->payload, message.data, 20);
    packet->checksum = get_checksum(packet);
    ++A.buffer_next;
    send_window();
}

/* called from layer 3, when a packet arrives for layer 4 */
void A_input(struct pkt packet) {
    if (packet.checksum != get_checksum(&packet)) {
        printf("  A_input: packet corrupted. drop.\n");
        return;
    }
    if (packet.acknum < A.base) {
        printf("  A_input: got NAK (ack=%d). drop.\n", packet.acknum);
        return;
    }
    printf("  A_input: got ACK (ack=%d)\n", packet.acknum);
    A.base = packet.acknum + 1;
    if (A.base == A.nextseq) {
        stoptimer(0);
        printf("  A_input: stop timer\n");
        send_window();
    } else {
        starttimer(0, A.estimated_rtt);
        printf("  A_input: timer + %f\n", A.estimated_rtt);
    }
}

/* called when A's timer goes off */
void A_timerinterrupt(void) {
    for (int i = A.base; i < A.nextseq; ++i) {
        struct pkt *packet = &A.packet_buffer[i % BUFSIZE];
        printf("  A_timerinterrupt: resend packet (seq=%d): %s\n", packet-
>seqnum, packet->payload);
        tolayer3(0, *packet);
    }
    starttimer(0, A.estimated_rtt);
    printf("  A_timerinterrupt: timer + %f\n", A.estimated_rtt);
}

/* the following routine will be called once (only) before any other */
/* entity A routines are called. You can use it to do any initialization */
void A_init(void) {
    A.base = 1;
    A.nextseq = 1;
```

```
    A.window_size = 8;
    A.estimated_rtt = 15;
    A.buffer_next = 1;
}

/* Note that with simplex transfer from a-to-B, there is no B_output() */

/* called from layer 3, when a packet arrives for layer 4 at B*/
void B_input(struct pkt packet) {
    if (packet.checksum != get_checksum(&packet)) {
        printf("  B_input: packet corrupted. send NAK (ack=%d)\n",
B.packet_to_send.acknum);
        tolayer3(1, B.packet_to_send);
        return;
    }
    if (packet.seqnum != B.expect_seq) {
        printf("  B_input: not the expected seq. send NAK (ack=%d)\n",
B.packet_to_send.acknum);
        tolayer3(1, B.packet_to_send);
        return;
    }

    printf("  B_input: recv packet (seq=%d): %s\n", packet.seqnum,
packet.payload);
    tolayer5(1, packet.payload);

    printf("  B_input: send ACK (ack=%d)\n", B.expect_seq);
    B.packet_to_send.acknum = B.expect_seq;
    B.packet_to_send.checksum = get_checksum(&B.packet_to_send);
    tolayer3(1, B.packet_to_send);

    ++B.expect_seq;
}

/* called when B's timer goes off */
void B_timerinterrupt(void) {
    printf("  B_timerinterrupt: B doesn't have a timer. ignore.\n");
}

/* the following rouytine will be called once (only) before any other */
/* entity B routines are called. You can use it to do any initialization */
void B_init(void) {
    B.expect_seq = 1;
    B.packet_to_send.seqnum = -1;
    B.packet_to_send.acknum = 0;
    memset(B.packet_to_send.payload, 0, 20);
    B.packet_to_send.checksum = get_checksum(&B.packet_to_send);
}
```