main.c 31 Share Run

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <arpa/inet.h>
#include <netinet/if_ether.h>
#include <sys/socket.h>
#include <unistd.h>

#define CACHE_SIZE 10

typedef struct {
    char ip[INET_ADDRSTRLEN];
    unsigned char mac[ETH_ALEN];
} arp_cache_entry;

arp_cache_entry arp_cache[CACHE_SIZE];

void add_to_cache(const char *ip, unsigned char *mac) {
    for (int i = 0; i < CACHE_SIZE; i++) {
        if (strlen(arp_cache[i].ip) == 0) {
            strcpy(arp_cache[i].ip, ip);
            memcpy(arp_cache[i].mac, mac, ETH_ALEN);
            break;
        }
    }
}
```

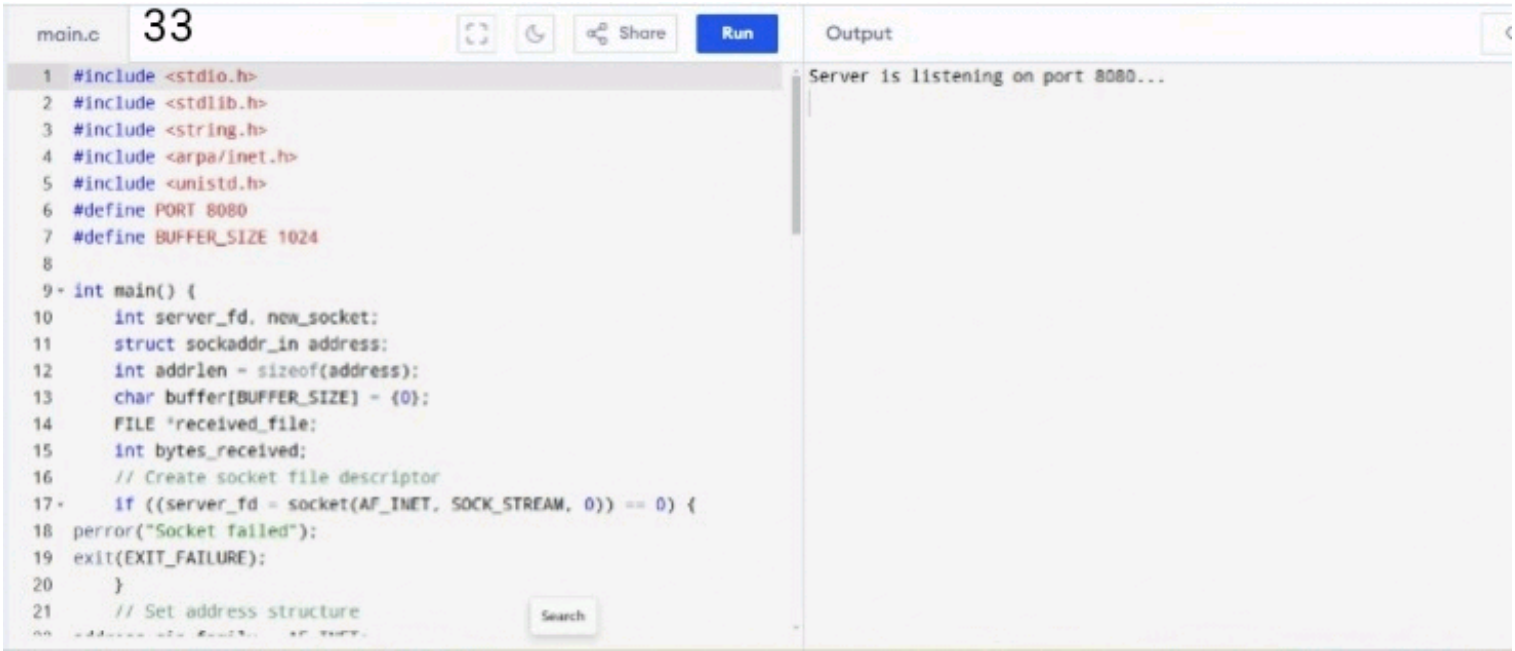
Output Clear

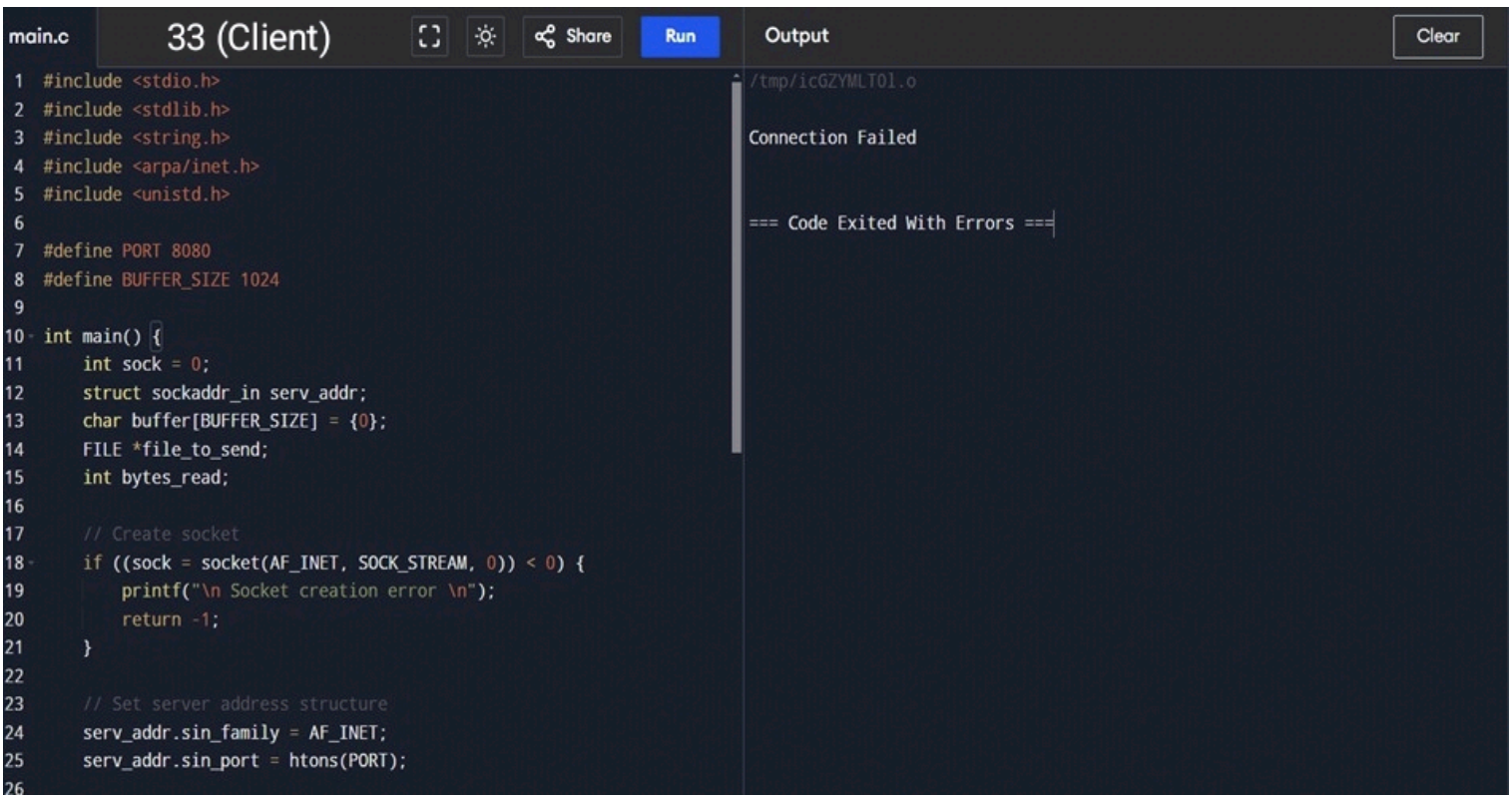
```
/tmp/8J6GoxASNj.o
IP: 192.168.1.1, MAC: 00:11:22:33:44:55


=== Code Execution Successful ===
```

main.c 32 Share Run Output Clear

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define FLAG_SEQUENCE "01111110"

// Function to perform bit stuffing
void bit_stuffing(const char *input, char *output) {
    int count = 0; // Count of consecutive '1's
    int j = 0; // Index for output array

    for (int i = 0; input[i] != '\0'; i++) {
        output[j++] = input[i]; // Copy the current bit to output

        // Count consecutive '1's
        if (input[i] == '1') {
            count++;
        } else {
            count = 0; // Reset count if '0' is found
        }

        // If we have five consecutive '1's, insert a '0'
        if (count == 5) {
            output[j++] = '0'; // Stuff a '0'
            count = 0; // Reset count after stuffing
```

```
/tmp/gudr89sSou.o
Input Data: 11111011101111101011111
Stuffed Data: 111110011101111101010111110
Destuffed Data: 11111011101111101011111


=== Code Execution Successful ===
```


main.c
33
Share
Run
Output
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <arpa/inet.h>
#include <unistd.h>
#define PORT 8080
#define BUFFER_SIZE 1024
int main() {
int server_fd, new_socket;
struct sockaddr_in address;
int addrlen = sizeof(address);
char buffer[BUFFER_SIZE] = {0};
FILE *received_file;
int bytes_received;
// Create socket file descriptor
if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
perror("Socket failed");
exit(EXIT_FAILURE);
}
// Set address structure
Search
Server is listening on port 8080...

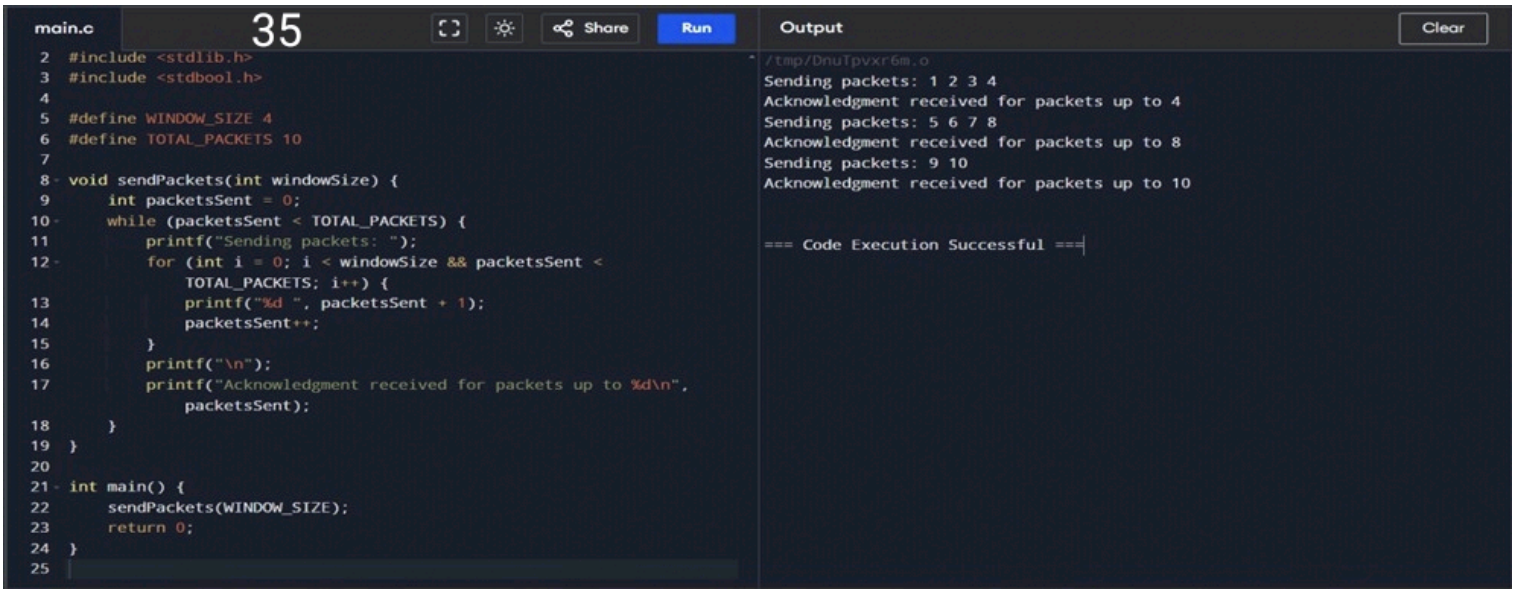

main.c
33 (Client)
Share
Run
Output
Clear
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <arpa/inet.h>
#include <unistd.h>
#define PORT 8080
#define BUFFER_SIZE 1024
int main() {
int sock = 0;
struct sockaddr_in serv_addr;
char buffer[BUFFER_SIZE] = {0};
FILE *file_to_send;
int bytes_read;
// Create socket
if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
printf("\n Socket creation error \n");
return -1;
}
// Set server address structure
serv_addr.sin_family = AF_INET;
serv_addr.sin_port = htons(PORT);
/tmp/icGZYMLTOl.o
Connection Failed
=== Code Exited With Errors ===

main.c 34 Share Run

```c
#include <stdio.h>
#include <string.h>

#define POLYNOMIAL 0x07

unsigned char compute_crc(const char *data) {
    unsigned char crc = 0; // Initial CRC value
    size_t len = strlen(data);

    for (size_t i = 0; i < len; i++) {
        crc ^= (data[i] & 0xFF);

        for (int j = 0; j < 8; j++) {
            if (crc & 0x80) {
                crc = (crc << 1) ^ POLYNOMIAL;
            } else {
                crc <<= 1;
            }
        }
    }
    return crc;
}

void simulate_error(char *data) {
    size_t len = strlen(data);
    if (len > 0) {
```

Output Clear

```
/tmp/GJirRrSc5d.o
ERROR!
Original Data: Hello, World!
Computed CRC: 0x87
Data after error: Iello, World!
Error detected in data!


=== Code Execution Successful ===
```


main.c
35
Share
Run
Output
Clear
#include <stdlib.h>
#include <stdbool.h>
#define WINDOW_SIZE 4
#define TOTAL_PACKETS 10
void sendPackets(int windowSize) {
int packetsSent = 0;
while (packetsSent < TOTAL_PACKETS) {
printf("Sending packets: ");
for (int i = 0; i < windowSize && packetsSent <
TOTAL_PACKETS; i++) {
printf("%d ", packetsSent + 1);
packetsSent++;
}
printf("\n");
printf("Acknowledgment received for packets up to %d\n",
packetsSent);
}
}
int main() {
sendPackets(WINDOW_SIZE);
return 0;
}
/tmp/DnuTpvxr6m.o
Sending packets: 1 2 3 4
Acknowledgment received for packets up to 4
Sending packets: 5 6 7 8
Acknowledgment received for packets up to 8
Sending packets: 9 10
Acknowledgment received for packets up to 10
=== Code Execution Successful ===