Computer Networks program, with easy-to-understand concepts, a line-by-line breakdown of the code, and viva questions.

---

1. Bit Stuffing Program (`WK1 Bit Stuffing.txt`)

Concept:

- **Bit Stuffing**: In data transmission, after every 5 consecutive '1' bits, a '0' bit is inserted to ensure that the receiver can distinguish between actual data and a flag sequence.
- **Bit Unstuffing**: At the receiver's end, the extra '0' bit is removed to retrieve the original data.

Code Explanation:

```
c
Copy code
void bit_stuffing(const char* input, char* stuffed) {
    int count = 0; // To count consecutive '1's
    int j = 0;
    for (int i = 0; i < strlen(input); i++) {
        stuffed[j++] = input[i]; // Copy data bit by bit
        if (input[i] == '1') {
            count++; // Increment count for '1'
            if (count == 5) { // If 5 consecutive '1's are encountered
                stuffed[j++] = '0'; // Insert a '0'
                count = 0; // Reset the count
            }
        } else {
            count = 0; // Reset count if bit is '0'
        }
    }
    stuffed[j] = '\0'; // Null-terminate stuffed data
}
```

- **Stuffing**: Every time there are 5 consecutive '1's, a '0' is inserted.

```
c
Copy code
void bit_unstuffing(const char* stuffed, char* unstuffed) {
    int count = 0, j = 0;
    for (int i = 0; i < strlen(stuffed); i++) {
        unstuffed[j++] = stuffed[i];
        if (stuffed[i] == '1') {
            count++;
            if (count == 5) { // Skip the stuffed '0' after 5 consecutive '1's
                i++;
                count = 0;
            }
        } else {
            count = 0;
        }
    }
    unstuffed[j] = '\0'; // Null-terminate unstuffed data
}
```

- **Unstuffing**: After detecting 5 consecutive '1's, the next bit is skipped (stuffed '0').

Viva Questions:

1. **What is Bit Stuffing?** Bit stuffing is the process of adding an extra '0' after five consecutive '1's to prevent misinterpretation of data.

2. **What problem does Bit Stuffing solve?** It helps distinguish data from control information, such as flags, during transmission.

---

2. Character Stuffing Program (`WK2 Character Stuffing.txt`)

Concept:

- **Character Stuffing**: In this technique, special flag or delimiter characters are inserted into the data to distinguish it from control characters. These extra characters are removed at the receiver's end.

Code Explanation:

```
c
Copy code
void char_stuffing(const char* input, char* stuffed, const char* flag, const char*
delimiter, int position) {
    int j = 0;
    for (int i = 0; delimiter[i] != '\0'; i++) {
        stuffed[j++] = delimiter[i]; // Add delimiter at the start
    }
    for (int i = 0; i < strlen(input); i++) {
        if (i == position) { // Add flag after 'position' characters
            for (int k = 0; flag[k] != '\0'; k++) {
                stuffed[j++] = flag[k];
            }
        }
        stuffed[j++] = input[i]; // Copy actual data
    }
    for (int i = 0; delimiter[i] != '\0'; i++) {
        stuffed[j++] = delimiter[i]; // Add delimiter at the end
    }
    stuffed[j] = '\0';
}
```

- **Stuffing**: Delimiters are added at the beginning and end, and a flag is inserted after a specified number of characters.

```
c
Copy code
void char_unstuffing(const char* stuffed, char* unstuffed, const char* flag, const
char* delimiter) {
    int skip = strlen(delimiter);
    for (int i = skip; i < strlen(stuffed) - skip; i++) {
        if (strncmp(&stuffed[i], flag, strlen(flag)) == 0) {
            i += strlen(flag) - 1; // Skip the flag
            continue;
        }
        unstuffed[j++] = stuffed[i];
    }
    unstuffed[j] = '\0'; // Null-terminate unstuffed data
}
```

- **Unstuffing**: Delimiters and flags are removed to recover the original data.

Viva Questions:

1. **What is Character Stuffing?** It is a method where extra characters (flags or delimiters) are added to the data for distinguishing control information from the actual data.

2. **Why are flags and delimiters used in Character Stuffing?** They are used to mark the beginning and end of a data frame, ensuring proper data transmission.

---

3. CRC-16 Program (`WK3 CRC-16.txt`)

Concept:

- **Cyclic Redundancy Check (CRC)**: A method used to detect errors in data transmission. It involves dividing the data by a polynomial (the key) and appending the remainder to the original data. The receiver repeats the division to check for errors.

Code Explanation:

```
c
Copy code
printf("Enter 16-bit Data (binary format): ");
gets(input);
printf("Enter Key (binary format): ");
gets(key);
```

- **Input**: The program accepts 16-bit data and a binary key (polynomial) from the user.

```
c
Copy code
strcpy(temp, input);
for (i = 0; i < keylen - 1; i++) {
    strcat(temp, "0"); // Append (keylen-1) zeros to data for CRC calculation
}
```

- **Appending Zeros**: Zeros are appended to the data to match the size required for division by the key.

```
c
Copy code
for (i = 0; i < msglen; i++) {
    if (quot[i] == '0') {
        for (j = 0; j < keylen; j++) {
            key[j] = '0'; // Use 0 if quotient bit is 0
        }
    } else {
        for (j = 0; j < keylen; j++) {
            key[j] = key1[j]; // Use original key if quotient bit is 1
        }
    }
    // Perform XOR for division (remainder calculation)
    for (j = keylen - 1; j > 0; j--) {
        rem[j - 1] = (temp[j] == key[j]) ? '0' : '1';
    }
    rem[keylen - 1] = temp[keylen + i]; // Bring down next bit
    strcpy(temp, rem); // Update temp with remainder
}
```

- **XOR Division**: XOR is used to divide the data by the key and calculate the remainder.

Viva Questions:

1. **What is CRC?** CRC (Cyclic Redundancy Check) is a method for detecting errors in data transmission by appending a calculated remainder (checksum) to the data.
2. **How does CRC-16 detect errors?** The sender appends the remainder to the data, and the receiver performs the same division. If the remainder is zero, the data is considered error-free.

---

4. Dijkstra's Algorithm Program (`WK4 Dijkstra.txt`)

Concept:

- **Dijkstra's Algorithm**: A shortest path algorithm that finds the shortest distance from a source node to all other nodes in a weighted graph.

Code Explanation:

```
c
Copy code
int G[MAX][MAX], i, j, n, u;
printf("\nEnter the number of vertices: ");
scanf("%d", &n);
```

- **Input**: The number of vertices and the adjacency matrix of the graph are input by the user.

```
c
Copy code
for (i = 0; i < n; i++) {
    for (j = 0; j < n; j++) {
        if (G[i][j] == 0) {
            cost[i][j] = INFINITY; // If no direct edge, set cost to INFINITY
        } else {
            cost[i][j] = G[i][j]; // Otherwise, set cost to the edge weight
        }
    }
}
```

- **Cost Calculation**: The adjacency matrix is converted to a cost matrix. A value of `0` means no direct connection, so the cost is set to infinity.

```
c
Copy code
while (count < n - 1) {
    for (i = 0; i < n; i++) {
        if (distance[i] < mindistance && !visited[i]) {
            mindistance = distance[i]; // Find the node with the smallest distance
            nextnode = i;
        }
    }
    visited[nextnode] = 1; // Mark the node as visited
    for (i = 0; i < n; i++) {
        if (!visited[i] && (mindistance + cost[nextnode][i] < distance[i])) {
            distance[i] = mindistance + cost[nextnode][i]; // Update distances
            pred[i] = nextnode; // Set predecessor
        }
    }
    count++;
}
```

- **Main Loop**: The algorithm iteratively selects the unvisited node with the shortest distance and updates the distances of its neighbors.

Viva Questions:

1. **What is Dijkstra's Algorithm?** Dijkstra's algorithm is a shortest path algorithm used to find the shortest distance between a source node and all other nodes in a graph.

2. **How does Dijkstra's Algorithm work?** It repeatedly selects the unvisited node with the smallest distance, updates the distances to its neighbors, and marks it as visited.

Wireshark Installation (`WK5 Wireshark Installation.pdf`)

Concept:

- **Wireshark**: A powerful, open-source network packet analyzer used to capture and analyze network traffic. It allows you to inspect data at a granular level, which is helpful for network troubleshooting, security analysis, and protocol debugging.

Installation Steps:

1. **Download Wireshark** from the official website Wireshark Download.
2. **Install Wireshark** by following the prompts and agreeing to the terms.
   - Select the default directory (`C:\Program Files\Wireshark`) and optional desktop/start menu shortcuts.
3. **Install WinPcap**: This is a required driver for Windows to capture network packets directly from the network interface. Follow the prompts to install it.
4. **Finish Installation**: Once WinPcap is installed, complete the Wireshark installation.

Features of Wireshark:

- **Live Packet Capture**: Captures data packets as they are transmitted over the network.
- **Detailed Protocol Information**: Displays network protocol details for each captured packet.
- **Filtering and Searching**: Allows filtering and searching through packets based on many criteria (e.g., IP address, port, protocol).
- **Statistics Generation**: Provides various statistics about the captured data, like load distribution and protocol hierarchy.

Viva Questions:

1. **What is Wireshark?** Wireshark is an open-source network packet analyzer used to capture and inspect network traffic.
2. **What is WinPcap, and why is it needed?** WinPcap is a packet capture driver that allows Wireshark to capture packets directly from the network interface in Windows.

---

2. Packet Capture Using Wireshark (`WK6 Wireshark.pdf`)

Concept:

- **Packet Capture**: Capturing network packets is essential for analyzing network traffic. Wireshark allows users to capture all incoming and outgoing data packets on a specific network interface.

Step-by-Step Procedure:

1. **Select Capture/Interfaces**: Open Wireshark and go to the `Capture` menu, then select `Interfaces`.
2. **Select the Interface**: Choose the network interface (e.g., Wi-Fi, Ethernet) where you want to capture packets.
3. **Start Capture**: Click the start button to begin capturing packets.

4. **Recreate the Problem**: Generate network traffic by performing actions like browsing or downloading to capture relevant packets.
5. **Stop the Capture**: Once the packets have been captured, click on the stop button.
6. **Save the Packet Trace**: Save the captured packets in the default format for future analysis.

Example Output:

```
plaintext
Copy code
Frame 1: 97 bytes on wire (776 bits), 97 bytes captured (776 bits)
Encapsulation type: Ethernet (1)
Source: 10.0.7.118, Destination: 224.0.0.251
Protocol: UDP
```

Viva Questions:

1. **What is packet capture?** Packet capture is the process of intercepting and recording data packets transmitted over a network.
2. **How does Wireshark capture packets?** Wireshark uses a network interface driver (WinPcap on Windows) to capture packets as they pass through the network adapter.

---

3. Viewing Captured Traffic (`WK7 Wireshark.pdf`)

Concept:

- **Viewing Traffic**: Once the packets are captured, they can be saved and opened later for offline analysis. Wireshark allows you to view and filter the captured data in great detail.

Step-by-Step Procedure:

1. **Capture Traffic**: Perform packet capture as in the previous experiment.
2. **Stop the Capture**: Once the traffic has been captured, stop the capture process.
3. **Save the Captured Data**: Save the captured packets to a file in the default format.
4. **Open Saved File**: Open the saved packet capture file in Wireshark.
5. **Analyze Traffic**: View the packets in Wireshark's detailed packet list pane.

Viva Questions:

1. **How can you view previously captured network traffic in Wireshark?** You can open a previously saved capture file from the `File` menu in Wireshark to view the traffic.
2. **What are the key details displayed for each packet in Wireshark?** Wireshark shows the source and destination addresses, protocol type, packet size, and additional protocol-specific information.

---

4. Simulate Statistics & Filters Using Wireshark (`WK8 Wireshark.pdf`)

Concept:

- **Filters and Statistics**: Wireshark allows users to apply display filters to focus on specific packets and generate various statistics (e.g., HTTP traffic distribution).

Step-by-Step Procedure:

1. **Select Filter Toolbar**: In Wireshark, click on the view menu and enable the filter toolbar.
2. **Find a Packet**: Use the `Find` function and type the desired filter (e.g., `http-host`) to display HTTP packets.
3. **Filter Packets**: Type the filter name and apply it to the captured packets.
4. **Generate Statistics**:
   - Go to the `Statistics` tab.
   - Click on `HTTP` and then select `Load Distribution`.
   - View the load distribution for HTTP traffic by host.

Example:

If the display filter `http-host` is applied, you can view HTTP requests filtered by the host name and analyze the load distribution.

Viva Questions:

1. **What is a display filter in Wireshark?** A display filter allows users to filter the captured packets based on specific criteria, like protocol, IP address, or port number.
2. **How can Wireshark's statistics be used for network analysis?** Wireshark generates statistics like load distribution, protocol hierarchy, and conversations that help in analyzing traffic patterns and identifying network issues.

This covers the detailed explanation and viva preparation for Wireshark