```
/*
-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.
-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.

Network Bridge Simulation with Store-and-Forward Algorithm
 *
 * This program simulates a network bridge that implements a store-
and-forward algorithm
 * to manage Ethernet frame forwarding based on MAC addresses. It
maintains a dynamic MAC
 * address table, learning source MAC addresses and their associated
ports, and forwards
 * frames to the appropriate port or broadcasts them if the
destination is unknown.
 *
 * Key Features:
 * - Learns MAC addresses and ports from incoming frames, updating a
linked list-based table.
 * - Forwards frames to a specific port if the destination MAC is
known, or broadcasts to all
 *   ports (except the sender's) if unknown.
 * - Removes table entries older than AGEING_TIME (300 seconds) to
prevent stale data.
 * - Validates input in the format:
"XX:XX:XX:XX:XX:XX,port,XX:XX:XX:XX:XX:XX" where
 *   MAC addresses use uppercase hexadecimal (0-9, A-F) and port is
a positive integer.
 *
 * Input:
 * - Reads input lines from stdin until "stop" is received.
 * - Expected format: MAC_sender,port,MAC_receiver (e.g.,
"00:1A:2B:3C:4D:5E,1000,00:1A:2B:3C:4D:5F").
 *
 * Output:
 * - Logs decisions (forwarding, broadcasting, or ignoring frames)
and MAC table state.
 *
 * Assumptions and Limitations:
 * - Input must be exactly 40 characters long (MACs, commas, and
port).
 * - MAC addresses use only uppercase hexadecimal digits (0-9, A-F).
 * - Invalid inputs (e.g., non-positive ports, invalid MACs) are
silently ignored.
 * - Memory allocation failures are not handled.
 *
 * Author: Francesco Piras
 * Date: October 6th 2025
 *
 *
-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.
-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-. */

#include <iostream>
#include <string.h>
```

```
#include <cstdlib>
#include <climits>
#include <ctime>
using namespace std;

#define AGEING_TIME 300

/* Definition of the Node struct */

struct Node {
    char MAC[19];
    int port;
    time_t timestamp;
    struct Node* next;
};

/* remove function: deletes THE NEXT NODE of the one passed to it,
and returns the next one */

void initialize(Node* current, char MAC[], int port) {
    strcpy(current->MAC, MAC);
    current->port = port;
    current->timestamp = time(NULL);
}

Node* remove(Node *current) {
    if (current == NULL || current->next == NULL) return current;
        Node *temp = current->next->next;
        delete current->next;
        current->next = temp;
        return current->next;
}

/* printList function: starting from the first node, PRINTS
ATTRIBUTES OF THE ENTIRE LIST and
DOES NOT RETURN A NODE */

void printList(Node* first) {
    Node* current = first;
    time_t now = time(NULL);
    cout << "[MAC TABLE STATE] | Timestamp: " << ctime(&(now));
    cout <<
"------------------------------------------------------------------
------" << endl;
    while (current != NULL) {
        cout << "MAC: " << current->MAC << " | ";
        cout << "Port: " << current->port << " | ";
        cout << "Timestamp: " << ctime(&(current->timestamp));
        current = current->next;
    }
    cout <<
"------------------------------------------------------------------
------" << endl << endl;
}
```

```
/* cleanList function: iterates through the list to check which node
exceeds 300 seconds
and should therefore be removed.
CASE 1: 'first' exceeds the time.
CASE 2: 'current' exceeds the time
(because the remove function removes current->next) */

Node* cleanList(Node* first) {
    if (first == nullptr) return nullptr;
    Node* current = first;

    if(first != NULL && (time(NULL) - first->timestamp) >
AGEING_TIME) {
        cout << "[MAINTENANCE] Cleaning MAC table: removing old
entities." << endl;
        cout << "[MAINTENANCE] Cleaning MAC table: removed 1 old
entity::MAC: " << first->MAC << endl;
        Node* temp = first;
        first = first->next;
        delete temp;
        current = first;
    }
    while (current != NULL && current->next != NULL) {
        if ((time(NULL) - current->next->timestamp) > AGEING_TIME) {
            cout << "[MAINTENANCE] Cleaning MAC table: removing old
entities." << endl;
            cout << "[MAINTENANCE] Cleaning MAC table: removed 1 old
entity::MAC: " << current->next->MAC << endl;
            current = remove(current);
        } else {
            current = current->next;
        }
    }

    if (first == nullptr) {
        cout << "[MAINTENANCE] MAC table is empty after cleaning."
<< endl;
    }

    time_t now = time(NULL);

    cout << "[MAINTENANCE] Cleaning MAC table: removing stale
entries older than 300 seconds." << endl;
    cout << "[MAINTENANCE RESULT] MAC table updated | Timestamp: "
<< ctime(&now) << endl;
    return first;
}

/* searchPort function: searches THROUGH THE ENTIRE LIST for a node
whose MAC_receiver
matches the MAC_sender of the current node.
If found, returns THE PORT of that node
through which the frame should be forwarded.
```

```
Otherwise, returns -1 */

int searchPort(Node* first, char MAC_receiver[], int values[]) {
    if (first-> next == NULL) return -2;
    Node* current = first;
    int i = 0;
    int port = 0;
    while (current != NULL) {
        if (strcmp(current->MAC, MAC_receiver) == 0) {
            port = current->port;
            return port;
        } else {
            values[i] = current->port;
            i++;
        }
    current = current->next;
    }
    return -1;
}

/* filtering function: decides the network output.
If MAC_sender and MAC_receiver are the same
(frame sent to itself), the request is ignored.
Otherwise, it searches for a match using the searchPort function.
If the port is found, the frame is sent to that port.
Otherwise, it's broadcasted to all ports of the bridge
except the caller's port */

void filtering(char MAC_sender[], int port, char MAC_receiver[],
Node* first, int values[]) {
    if(strcmp(MAC_sender, MAC_receiver) == 0) {
        cout << "[FILTERING DECISION] Frame ignored - source and
destination MAC addresses identical." << endl;
    } else {
        for (int i = 0; i < 100; i++) {
            values[i] = 0;
        }
        int p = searchPort(first, MAC_receiver, values);
        if (p >= 0) {
            cout << "[FILTERING DECISION] Forwarding frame to port "
<< p << endl;
        } else if (p == -1) {
            cout << "[FILTERING DECISION] Destination MAC not found.
Broadcasting frame to all ports: ";
            for(int i=0; i<100; i++) {
                if (values[i] == 0) break;
                if (values[i] != port) {
                    cout << values[i] << " ";
                }
            }
            cout << endl;
        } else {
            cout << "[FILTERING DECISION] No port to broadcast to"
<< endl;
```

```
        }
    }
}

/* learning function: stores nodes in the dynamic list */

Node* learning(char MAC_sender[], int port, Node* first, bool
&IsEmpty) {

    /* Brief check for empty list */
    if (IsEmpty) {
        /* NOW creates a node and assigns it certain values */
        first = new Node();
        initialize(first, MAC_sender, port);
        first->next = nullptr;
        IsEmpty = false;
        return first;
    }

    Node* current = first;

    /* Looks for matching MAC addresses */
    while (current != NULL) {
        if (strcmp(current->MAC, MAC_sender) == 0) {
            /* If only the MAC matches, update the port */
            if (current->port != port) {
                current->port = port;
            }
            /* Always update the timestamp */
            current->timestamp = time(NULL);
            return first;
        }
        if (current->next == NULL) {
            break;
        }
        current = current->next;
    }

    current->next = new Node();
    current = current->next;
    initialize(current, MAC_sender, port);

    return first;
}

int main() {
    time_t now = time(NULL);
    cout << "[BRIDGE MODULE] Initializing Store-and-Forward
Algorithm..." << endl << "[BRIDGE MODULE] Learning Table Initialized
| Timestamp: " << ctime(&now) << endl;
    Node* first = nullptr;

    bool IsEmpty = true;
    Node* current = first;
    char tokens[40];
    char MAC_sender[18];
    int port;
    char MAC_receiver[18];
    int values[100] = {};

    do {
        cin.getline(tokens, 43);

        if (cin.fail()) {
            cin.clear();
            cin.ignore(INT_MAX, '\n');
            continue;
        }
        cout << endl << endl;

        if (strcmp(tokens, "stop") == 0) {
            break;
        }

        if (tokens[17] != ',' || tokens[22] != ',' ||
strlen(tokens) != 40) continue;
        char* token = strtok(tokens, ",");
        if (token == NULL) continue;
        strcpy(MAC_sender, token);
        token = strtok(NULL, ",");
        if (token == NULL) continue;
        port = atoi(token);
        token = strtok(NULL, ",");
        if (token == NULL) continue;
        strcpy(MAC_receiver, token);
        bool error = false;
        for (int i = 0; i < 17; i++) {
            if (i % 3 == 2) {
                if (MAC_sender[i] != ':' || MAC_receiver[i] != ':')
error = true; continue;
            } else {
                if (!(MAC_sender[i] >= '0' && MAC_sender[i] <= 'F')
|| !(MAC_receiver[i] >= '0' && MAC_receiver[i] <= 'F')) error =
true; continue;
            }
        }
        if (error) continue;

        first = learning(MAC_sender, port, first, IsEmpty);
        filtering(MAC_sender, port, MAC_receiver, first, values);

        first = cleanList(first);
        printList(first);
    } while (strcmp(tokens, "stop") != 0);

    return 0;
}
```

```
/*
-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.
-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.

                                            OUTPUT EXAMPLE:

[BRIDGE MODULE] Initializing Store-and-Forward Algorithm...
[BRIDGE MODULE] Learning Table Initialized | Timestamp: Mon Oct  6
17:36:28 2025

00:1A:2B:3C:4D:5E,1000,01:1A:2B:3C:4D:5E

[FILTERING DECISION] No port to broadcast to
[MAINTENANCE] Cleaning MAC table: removing stale entries older than
300 seconds.
[MAINTENANCE RESULT] MAC table updated | Timestamp: Mon Oct  6
17:37:02 2025

[MAC TABLE STATE] | Timestamp: Mon Oct  6 17:37:02 2025
--------------------------------------------------------------------
-----
MAC: 00:1A:2B:3C:4D:5E | Port: 1000 | Timestamp: Mon Oct  6 17:37:02
2025
--------------------------------------------------------------------
-----

02:1A:2B:3C:4D:5E,2000,01:1A:2B:3C:4D:5E

[FILTERING DECISION] Destination MAC not found. Broadcasting frame
to all ports: 1000
[MAINTENANCE] Cleaning MAC table: removing stale entries older than
300 seconds.
[MAINTENANCE RESULT] MAC table updated | Timestamp: Mon Oct  6
17:37:38 2025

[MAC TABLE STATE] | Timestamp: Mon Oct  6 17:37:38 2025
--------------------------------------------------------------------
-----
MAC: 00:1A:2B:3C:4D:5E | Port: 1000 | Timestamp: Mon Oct  6 17:37:02
2025
MAC: 02:1A:2B:3C:4D:5E | Port: 2000 | Timestamp: Mon Oct  6 17:37:38
2025
--------------------------------------------------------------------
-----

03:1A:2B:3C:4D:5E,3000,03:1A:2B:3C:4D:5E

[FILTERING DECISION] Frame ignored - source and destination MAC
addresses identical.
[MAINTENANCE] Cleaning MAC table: removing stale entries older than
300 seconds.
[MAINTENANCE RESULT] MAC table updated | Timestamp: Mon Oct  6
17:38:12 2025
```

```
[MAC TABLE STATE] | Timestamp: Mon Oct  6 17:38:12 2025
--------------------------------------------------------------------
-----
MAC: 00:1A:2B:3C:4D:5E | Port: 1000 | Timestamp: Mon Oct  6 17:37:02
2025
MAC: 02:1A:2B:3C:4D:5E | Port: 2000 | Timestamp: Mon Oct  6 17:37:38
2025
MAC: 03:1A:2B:3C:4D:5E | Port: 3000 | Timestamp: Mon Oct  6 17:38:12
2025
--------------------------------------------------------------------
-----

-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.
-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-. */
```