Project Report

On

USAGE OF ADDRESS RESOLUTION PROTOCOL

Submitted in partial fulfilment of the requirements for the award of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE & ENGINEERING

(Artificial Intelligence & Machine Learning)

by

Ms. V HARSHITIA - (22WH1A6602)

Ms. N JIJNASA - (22WH1A6635)

Ms. N PRASANNA - (22WH1A6643)

Ms. M SIRI CHANDANA - (22WH1A6661)

Under the esteemed guidance of

Ms. P Anusha

Assistant Professor, CSE(AI&ML)

BVRIT HYDERABAD College of Engineering for Women

(UGC Autonomous Institution | Approved by AICTE | Affiliated to JNTUH)

(NAAC Accredited - A Grade | NBA Accredited B.Tech. (EEE, ECE, CSE and IT)

Bachupally, Hyderabad – 500090

2024-25

ABSTRACT

The Address Resolution Protocol (ARP) is a crucial networking protocol used to map Internet Protocol (IP) addresses to Media Access Control (MAC) addresses within a local network. The ARP tool provides a practical simulation of ARP functionalities, enabling users to understand how IP-to-MAC address resolution works in real-world scenarios. This tool offers key operations such as ARP table initialization, entry addition, MAC address lookup, and ARP request simulation.

By interacting with this tool, users can visualize ARP request and response processes, explore scenarios like cache management and entry conflicts, and handle full-table conditions. The tool is designed to be user-friendly and educative, serving both networking students and professionals aiming to strengthen their knowledge of network communication at the data link layer. Its practical use extends to network troubleshooting, ensuring efficient IP-MAC mappings, and fostering a deeper comprehension of network infrastructure.

1. PROBLEM STATEMENT

In network communication, seamless interaction between devices requires accurate mapping between Internet Protocol (IP) addresses and Media Access Control (MAC) addresses. While IP addresses are used for logical routing across networks, MAC addresses are essential for physical data transmission at the data link layer within a local area network (LAN). The Address Resolution Protocol (ARP) is designed to bridge this gap by dynamically resolving IP addresses to corresponding MAC addresses. Despite its critical role in network communication, understanding the ARP mechanism is often challenging for students, network administrators, and IT professionals due to its underlying complexity and lack of direct visibility during regular network operations.

The core problem lies in the absence of an accessible, interactive learning platform that demonstrates how ARP works in real-world scenarios, including how it handles various network conditions like cache overflow, stale entries, and resolution failures. Additionally, there is a need for a practical tool that aids in network troubleshooting by simulating ARP behavior and demonstrating its impact on network performance.

Objective: To develop a comprehensive ARP simulation tool that provides an interactive environment for users to understand and experiment with the ARP process. The tool should enable users to:

- **Initialize and manage an ARP table** with a predefined capacity.
- **Add and remove ARP entries** dynamically, mimicking real-world scenarios.
- **Perform MAC address lookups** by entering IP addresses and observing successful or failed resolutions.
- **Simulate ARP requests and responses** to illustrate how devices query and receive MAC addresses within a LAN.
- **Handle edge cases** such as full ARP tables, invalid entries, and non-existent IP addresses, providing meaningful feedback and error handling.
- **Display the ARP table** in a user-friendly format to visualize IP-to-MAC mappings and entry status.

2. FUNCTIONAL REQUIREMENTS

1. ARP Table Management

- The tool shall initialize an ARP table with a fixed maximum capacity (e.g., 10 entries).
- The tool shall allow adding new ARP entries by specifying an IP address and a MAC address.
- The tool shall mark each ARP entry as valid or invalid based on its current state.
- The tool shall prevent adding new entries when the ARP table is full and display an appropriate error message.
- The tool shall allow removing or invalidating specific ARP entries.
- The tool shall display the current state of the ARP table, including IP addresses, MAC addresses, and validity status.

2. ARP Lookup and Resolution

- The tool shall allow users to input an IP address to look up its corresponding MAC address in the ARP table.
- The tool shall return the MAC address if the IP address is found in the table.
- The tool shall return an appropriate message if the IP address is not found or the entry is invalid.

3. ARP Request Simulation

- The tool shall simulate ARP requests for a given IP address.
- The tool shall display whether the MAC address was successfully resolved or not.
- The tool shall notify users when an ARP request fails due to a missing entry in the ARP table.

4. User Interaction and Feedback

- The tool shall provide clear and concise feedback for all user actions, including successful and failed operations.
- The tool shall display appropriate error messages for invalid inputs (e.g., incorrect IP or MAC address formats).
- The tool shall offer a user-friendly command-line interface for input and output operations.

5. Data Validation

- The tool shall validate IP addresses to ensure they conform to standard IPv4 format (e.g., 192.168.1.1).
- The tool shall validate MAC addresses to ensure they follow the standard format (e.g., 00:1A:2B:3C:4D:5E).
- The tool shall reject duplicate IP addresses and notify the user accordingly.

6. Display and Reporting

- The tool shall display the ARP table in a tabular format with columns for IP address, MAC address, and entry status.
- The tool shall provide a summary report of successful and failed ARP lookups and requests.

7. Error Handling

- The tool shall handle errors gracefully and provide meaningful feedback to guide users in correcting their actions.
- The tool shall prevent system crashes due to invalid or unexpected user inputs.

8. System Requirements

- The tool shall run as a standalone console application.
- The tool shall be compatible with standard C compilers and should not require external libraries for execution.

3. NON-FUNCTIONAL REQUIREMENTS

1. Performance

- The application should introduce minimal system overhead, ensuring efficient use of CPU and memory resources.
- The update interval for network statistics should be within **1-2 seconds** for the console and **1.5 seconds** for the GUI, maintaining responsiveness without excessive system resource consumption.
- The tool should handle continuous updates over extended periods without noticeable degradation in performance.

2. Usability

- The GUI should be user-friendly, with a simple, intuitive layout that allows users to easily monitor network statistics without prior technical knowledge.
- The console version should present data in a clear, well-formatted manner, with clear separation between different metrics to enhance readability.
- Tooltips and labels in the GUI should guide users for better interaction, and error messages should be clear and actionable.

3. Reliability

- The system must reliably fetch and display accurate network statistics, ensuring consistency between data points in both GUI and console modes.
- The application should handle edge cases (e.g., network counter wraparounds or unavailable network interfaces) without crashing or providing incorrect data.
- The system should have a high uptime and handle network interruptions gracefully, providing fallback messages when data is temporarily unavailable.

4. Portability

- The application should run seamlessly on **Windows, Linux, and macOS** platforms with minimal or no modifications.
- It should rely on widely available Python libraries such as **psutil** for system monitoring and **tkinter** for the GUI, ensuring easy setup across platforms.
- The tool should be compatible with Python versions **3.7 and above** to maximize its portability across different systems.

5. Scalability

- The application should support future enhancements to monitor multiple network interfaces simultaneously without significant code changes.
- It should be capable of handling larger deployments, such as server environments, where multiple instances can run concurrently for monitoring purposes.
- The tool should allow future extensions to incorporate additional metrics or integrate with external monitoring systems (e.g., Prometheus).

6. Maintainability

- The codebase should be well-documented, with clear comments explaining functions, modules, and workflows.
- The application should follow a modular architecture, separating concerns like data fetching, UI rendering, and formatting, making the system easy to maintain and extend.
- The code should adhere to standard Python coding conventions (e.g., **PEP 8**) for consistency and readability.

4. SOURCE CODE

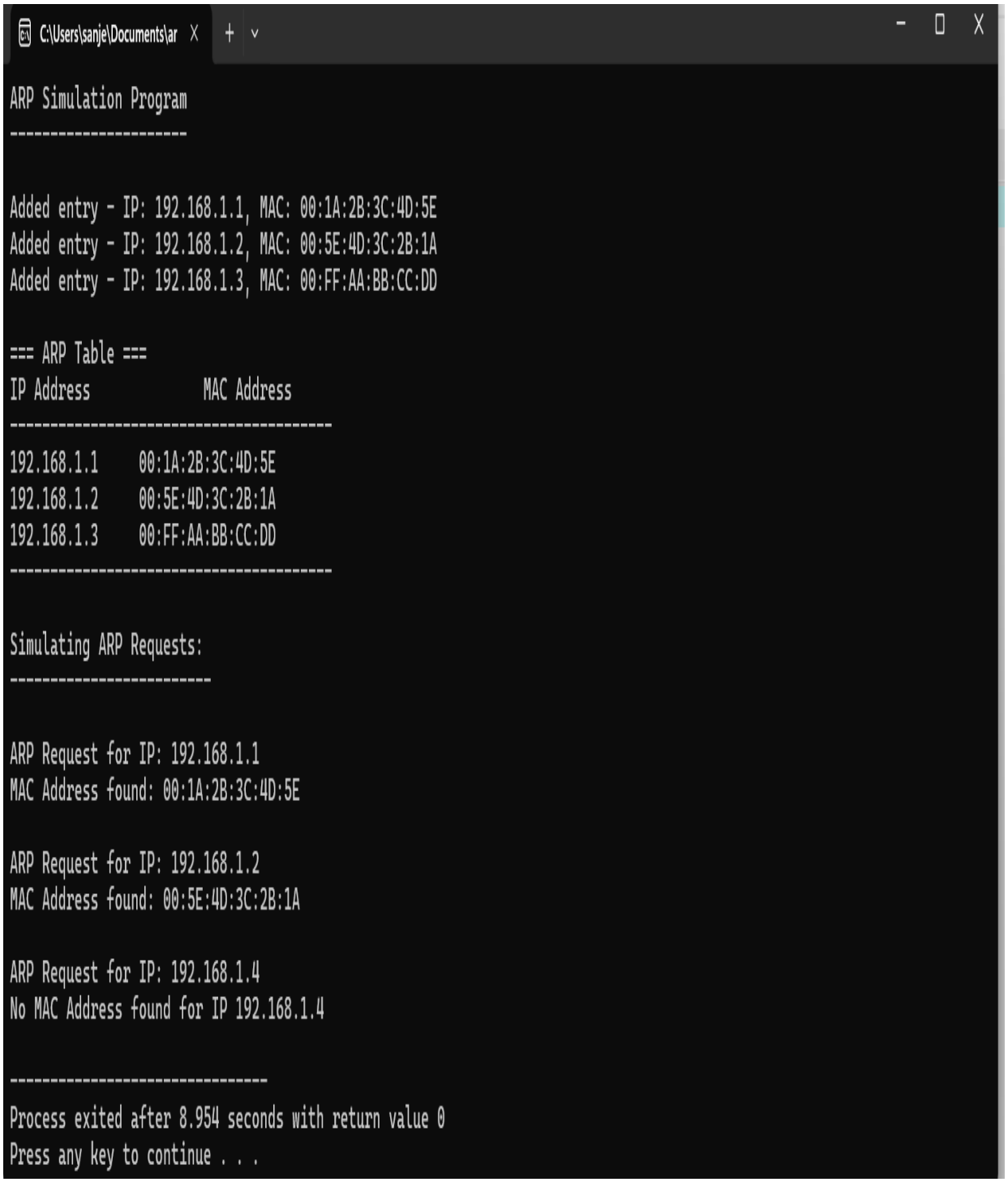
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_ENTRIES 10
// Structure to represent an ARP table entry
struct arp_entry {
    char ip_address[16];
    char mac_address[18];
    int is_valid;
};
// ARP table
struct arp_entry arp_table[MAX_ENTRIES];
int arp_entry_count = 0;
// Function to initialize the ARP table
void init_arp_table() {
    for (int i = 0; i < MAX_ENTRIES; i++) {
        arp_table[i].is_valid = 0;
    }
}
// Function to add an entry to the ARP table
void add_arp_entry(const char* ip, const char* mac) {
    if (arp_entry_count >= MAX_ENTRIES) {
        printf("ARP table is full!\n");
        return;
    }
```

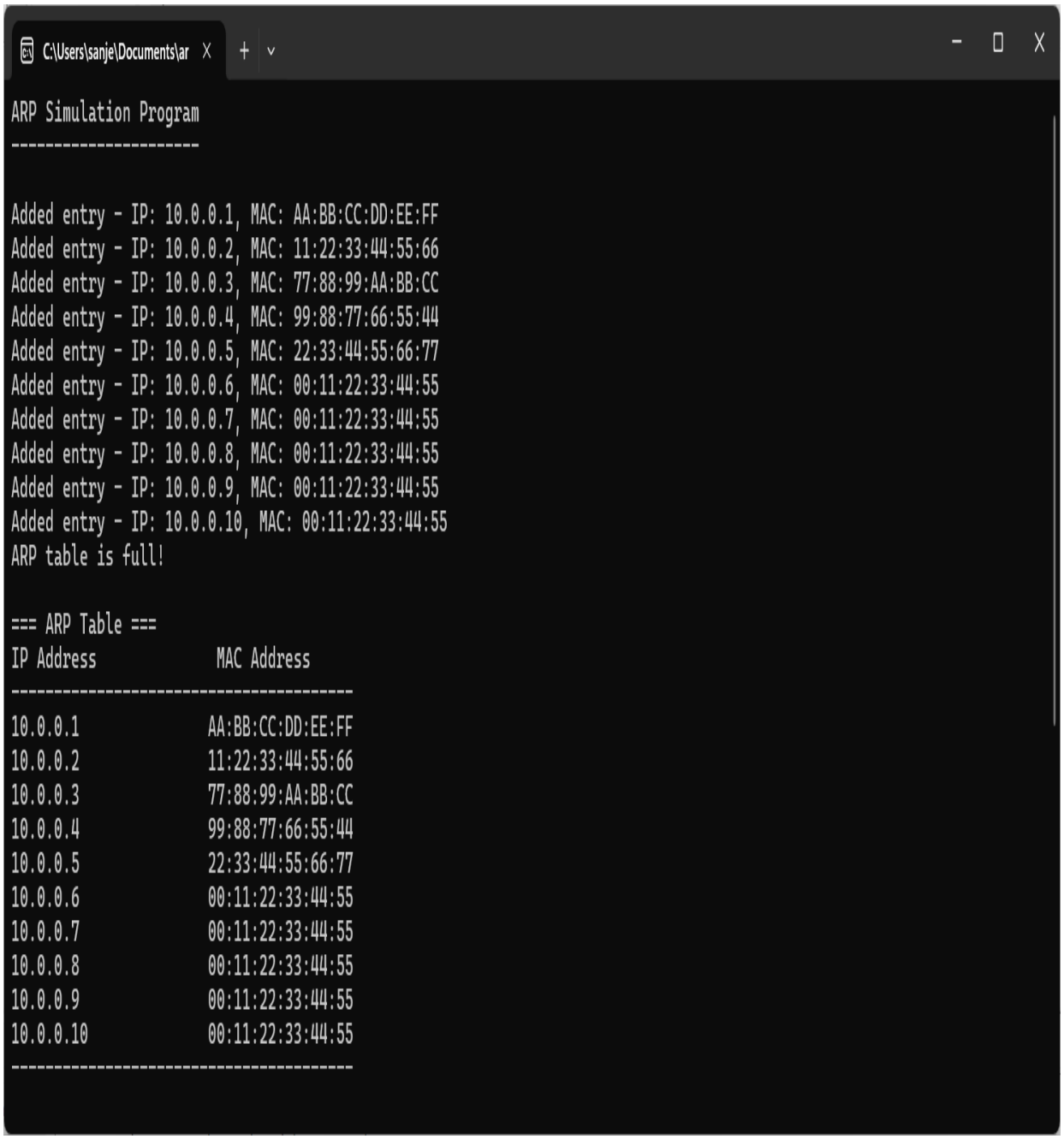
```
    strcpy(arp_table[arp_entry_count].ip_address, ip);
    strcpy(arp_table[arp_entry_count].mac_address, mac);
    arp_table[arp_entry_count].is_valid = 1;
    arp_entry_count++;
    printf("Added entry - IP: %s, MAC: %s\n", ip, mac);
}
// Function to look up MAC address for given IP
const char* lookup_mac_address(const char* ip) {
    for (int i = 0; i < arp_entry_count; i++) {
        if (arp_table[i].is_valid && strcmp(arp_table[i].ip_address, ip) == 0) {
            return arp_table[i].mac_address;
        }
    }
    return NULL;
}
// Function to display the ARP table
void display_arp_table() {
    printf("\n=== ARP Table ===\n");
    printf("IP Address\t\tMAC Address\n");
    printf("-----------------------------------------\n");
        for (int i = 0; i < arp_entry_count; i++) {
        if (arp_table[i].is_valid) {
            printf("%s\t%s\n",
                   arp_table[i].ip_address,
                  arp_table[i].mac_address);
        }
```

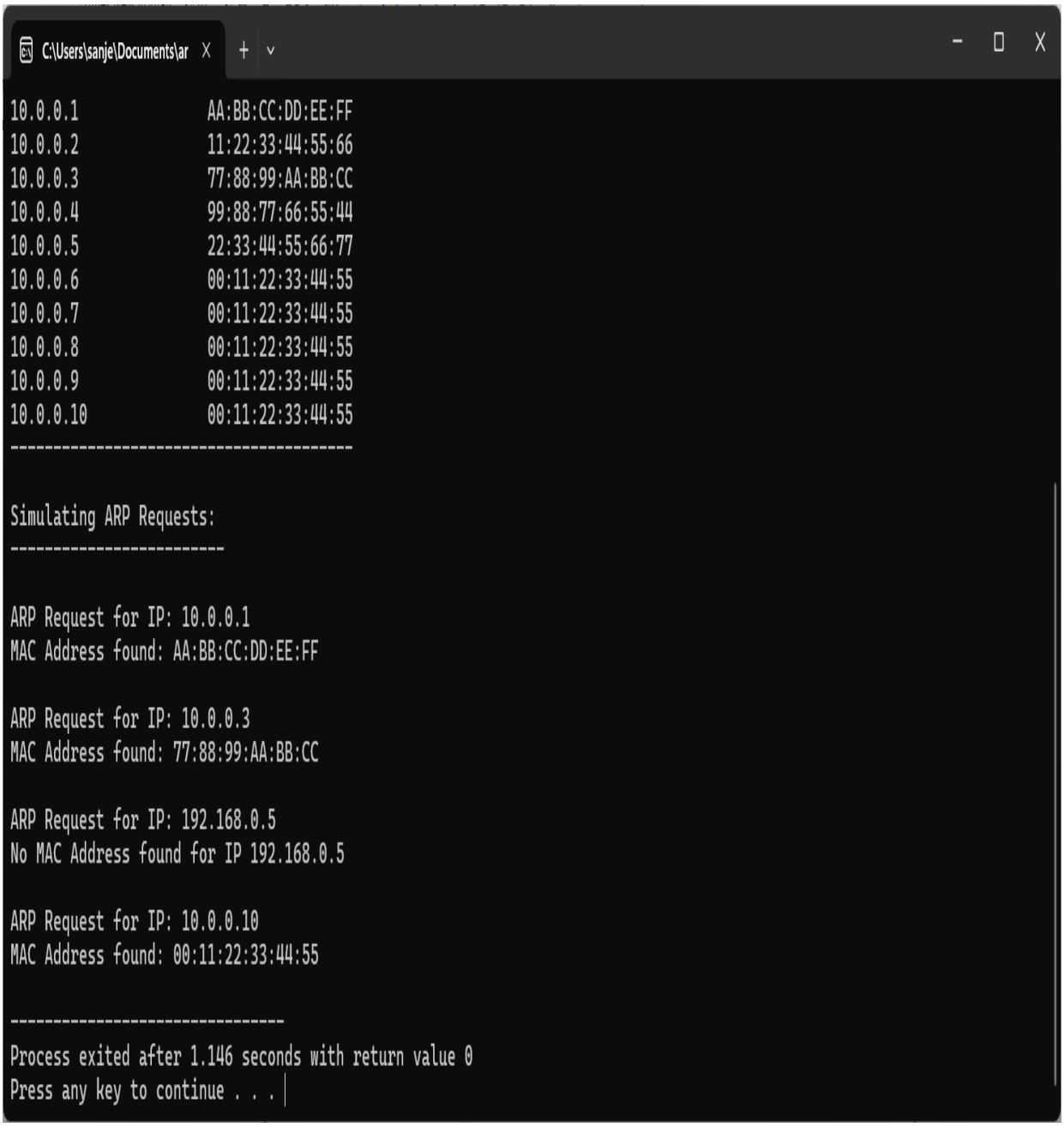
```
}
    printf("----------------------------------------\n");
}
// Function to simulate ARP request
void arp_request(const char* ip) {
    printf("\nARP Request for IP: %s\n", ip);
    const char* mac = lookup_mac_address(ip);

    if (mac) {
        printf("MAC Address found: %s\n", mac);
    } else {
        printf("No MAC Address found for IP %s\n", ip);
    }
}

int main() {
    // Initialize ARP table
    init_arp_table();

    printf("ARP Simulation Program\n");
    printf("----------------------\n\n");

    // Add some sample entries
    add_arp_entry("192.168.1.1", "00:1A:2B:3C:4D:5E");
    add_arp_entry("192.168.1.2", "00:5E:4D:3C:2B:1A");
    add_arp_entry("192.168.1.3", "00:FF:AA:BB:CC:DD");
```

```
    // Display the current ARP table
    display_arp_table();

    // Simulate some ARP requests
    printf("\nSimulating ARP Requests:\n");
    printf("------------------------\n");
    arp_request("192.168.1.1");
    arp_request("192.168.1.2");
    arp_request("192.168.1.4"); // This IP is not in the table

    return 0;
}
```

OUTPUT


C:\Users\sanje\Documents\ar
ARP Simulation Program
Added entry - IP: 192.168.1.1, MAC: 00:1A:2B:3C:4D:5E
Added entry - IP: 192.168.1.2, MAC: 00:5E:4D:3C:2B:1A
Added entry - IP: 192.168.1.3, MAC: 00:FF:AA:BB:CC:DD
=== ARP Table ===
IP Address MAC Address
192.168.1.1 00:1A:2B:3C:4D:5E
192.168.1.2 00:5E:4D:3C:2B:1A
192.168.1.3 00:FF:AA:BB:CC:DD
Simulating ARP Requests:
ARP Request for IP: 192.168.1.1
MAC Address found: 00:1A:2B:3C:4D:5E
ARP Request for IP: 192.168.1.2
MAC Address found: 00:5E:4D:3C:2B:1A
ARP Request for IP: 192.168.1.4
No MAC Address found for IP 192.168.1.4
Process exited after 8.954 seconds with return value 0
Press any key to continue . . .


C:\Users\sanje\Documents\ar
ARP Simulation Program
Added entry - IP: 10.0.0.1, MAC: AA:BB:CC:DD:EE:FF
Added entry - IP: 10.0.0.2, MAC: 11:22:33:44:55:66
Added entry - IP: 10.0.0.3, MAC: 77:88:99:AA:BB:CC
Added entry - IP: 10.0.0.4, MAC: 99:88:77:66:55:44
Added entry - IP: 10.0.0.5, MAC: 22:33:44:55:66:77
Added entry - IP: 10.0.0.6, MAC: 00:11:22:33:44:55
Added entry - IP: 10.0.0.7, MAC: 00:11:22:33:44:55
Added entry - IP: 10.0.0.8, MAC: 00:11:22:33:44:55
Added entry - IP: 10.0.0.9, MAC: 00:11:22:33:44:55
Added entry - IP: 10.0.0.10, MAC: 00:11:22:33:44:55
ARP table is full!
=== ARP Table ===
IP Address MAC Address
10.0.0.1 AA:BB:CC:DD:EE:FF
10.0.0.2 11:22:33:44:55:66
10.0.0.3 77:88:99:AA:BB:CC
10.0.0.4 99:88:77:66:55:44
10.0.0.5 22:33:44:55:66:77
10.0.0.6 00:11:22:33:44:55
10.0.0.7 00:11:22:33:44:55
10.0.0.8 00:11:22:33:44:55
10.0.0.9 00:11:22:33:44:55
10.0.0.10 00:11:22:33:44:55


C:\Users\sanje\Documents\ar
10.0.0.1 AA:BB:CC:DD:EE:FF
10.0.0.2 11:22:33:44:55:66
10.0.0.3 77:88:99:AA:BB:CC
10.0.0.4 99:88:77:66:55:44
10.0.0.5 22:33:44:55:66:77
10.0.0.6 00:11:22:33:44:55
10.0.0.7 00:11:22:33:44:55
10.0.0.8 00:11:22:33:44:55
10.0.0.9 00:11:22:33:44:55
10.0.0.10 00:11:22:33:44:55
Simulating ARP Requests:
ARP Request for IP: 10.0.0.1
MAC Address found: AA:BB:CC:DD:EE:FF
ARP Request for IP: 10.0.0.3
MAC Address found: 77:88:99:AA:BB:CC
ARP Request for IP: 192.168.0.5
No MAC Address found for IP 192.168.0.5
ARP Request for IP: 10.0.0.10
MAC Address found: 00:11:22:33:44:55
Process exited after 1.146 seconds with return value 0
Press any key to continue . . .