Command Line Argument, Variable Argument Handling Assignment:

1. Write a program to
   a. read a name(of max length 40 characters), ip address (as char * string in dotted notation) and port number (unsigned short) of the cloud server as command line arguments.
   b. Validate if the required number of arguments have been received before proceeding. Else report error and return.
   c. Validate every argument received for valid range of values.
      [Refer ip address range, port range to do validations]
   d. Store the values in a data structure and display using a function passing data structure
      void display(struct server *servercfg);
   e. Implement a function update() to prompt user, to modify all the server attributes and to display the updated configuration.
      // to read, update configuration and return status as SUCCESS/FAILURE

      Int update(struct server *servercfg);

   f. Specify atleast 6 test cases (positive and negative) to test command line inputs and update operations
   g. Check for memory leaks and fix them.

Ans:

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

#define MAX_NAME_LENGTH 40
#define MAX_IP_LENGTH 16

struct server {
   char name[MAX_NAME_LENGTH];   // Server name
   char ip[MAX_IP_LENGTH];       // IP address in dotted notation
   unsigned short port;          // Port number
};

void display(struct server *servercfg);
int update(struct server *servercfg);
int is_valid_ip(const char *ip);
int is_valid_port(unsigned short port);

int main(int argc, char *argv[]) {
    if (argc != 4) {
        printf("Error: Invalid number of arguments.\n");
        printf("Usage: %s <name> <ip_address> <port>\n", argv[0]);
        return 1;
    }

    struct server servercfg;

    // Read and validate name
    if (strlen(argv[1]) >= MAX_NAME_LENGTH) {
        printf("Error: Name length exceeds the maximum allowed (40
characters).\n");
        return 1;
    }
    strncpy(servercfg.name, argv[1], MAX_NAME_LENGTH - 1);
    servercfg.name[MAX_NAME_LENGTH - 1] = '\0';

    // Read and validate IP address
    if (!is_valid_ip(argv[2])) {
        printf("Error: Invalid IP address.\n");
        return 1;
    }
    strncpy(servercfg.ip, argv[2], MAX_IP_LENGTH - 1);
    servercfg.ip[MAX_IP_LENGTH - 1] = '\0';

    // Read and validate port number
    unsigned short port = (unsigned short)atoi(argv[3]);
    if (!is_valid_port(port)) {
        printf("Error: Invalid port number. It should be between 1 and
65535.\n");
        return 1;
    }
    servercfg.port = port;

    // Display the initial configuration
    display(&servercfg);

    // Prompt user to update the server configuration
    if (update(&servercfg) == 1) {
        display(&servercfg);
    } else {
        printf("Update failed.\n");
    }

    return 0;
}

// Function to display server configuration
```

```
void display(struct server *servercfg) {
    printf("\nServer Configuration:\n");
    printf("Name: %s\n", servercfg->name);
    printf("IP Address: %s\n", servercfg->ip);
    printf("Port: %u\n", servercfg->port);
}

// Function to update server configuration
int update(struct server *servercfg) {
    char input[MAX_NAME_LENGTH];
    char ip[MAX_IP_LENGTH];
    unsigned short port;

    // Prompt for server name
    printf("\nEnter new server name (max 40 chars): ");
    fgets(input, MAX_NAME_LENGTH, stdin);
    input[strcspn(input, "\n")] = '\0'; // Remove newline character
    if (strlen(input) > 0) {
        strncpy(servercfg->name, input, MAX_NAME_LENGTH - 1);
        servercfg->name[MAX_NAME_LENGTH - 1] = '\0';
    }

    // Prompt for IP address
    printf("Enter new IP address: ");
    fgets(ip, MAX_IP_LENGTH, stdin);
    ip[strcspn(ip, "\n")] = '\0'; // Remove newline character
    if (is_valid_ip(ip)) {
        strncpy(servercfg->ip, ip, MAX_IP_LENGTH - 1);
        servercfg->ip[MAX_IP_LENGTH - 1] = '\0';
    } else {
        printf("Invalid IP address format.\n");
        return 0;
    }

    // Prompt for port number
    printf("Enter new port number (1-65535): ");
    if (scanf("%hu", &port) != 1 || !is_valid_port(port)) {
        printf("Invalid port number.\n");
        return 0;
    }
    servercfg->port = port;

    return 1; // Success
}

// Function to validate IP address format (in dotted notation)
int is_valid_ip(const char *ip) {
    int dots = 0;
    int num;
    int i = 0;

    while (*ip) {
        if (isdigit(*ip)) {
            num = num * 10 + (*ip - '0');
        } else if (*ip == '.') {
            if (num < 0 || num > 255) {
                return 0;
            }
            num = 0;
            dots++;
        } else {
            return 0;
        }
        ip++;
    }

    return (dots == 3 && num >= 0 && num <= 255);
}

// Function to validate port number
int is_valid_port(unsigned short port) {
    return (port >= 1 && port <= 65535);
}
```

2. Implement a log() with signature as below to display all the input arguments as per their type. [Hint: In log() , use vfprintf() to display the received inputs]

```
void log(const char *format, …);

For e.g.

int main()
{
                    int count = 10;
                    char prefix  =  'h';
                    char label[] = "India";
                    …
                    log("count:%d, prefix:%c, label:%s", count, prefix,
label);
                    …
}
```

Expected Output:

```
          count:10,prefix:h,label:India
```

ans:

```
#include <stdio.h>
#include <stdarg.h>
```

```
void log(const char *format, ...) {
    va_list args;         // Declare a va_list to store the variable arguments
    va_start(args, format);  // Initialize the va_list with the format string

    vfprintf(stdout, format, args);  // Output the formatted string to stdout

    va_end(args);  // Clean up the va_list after use
}

int main() {
    int count = 10;
    char prefix = 'h';
    char label[] = "India";

    // Calling log() with format specifiers for integer, character, and string
    log("count:%d, prefix:%c, label:%s\n", count, prefix, label);

    return 0;
}
```

3. Refer the code "find_max.c". Add a function below to accept variable number of strings and to return the string with maximum length to the caller. In case of strings with same length, return the first string in the input sequence

```
max_len_string(<variable number of arguments>)

Eg. Code below shoud output “hello”
char *ptr = max_len_string(“hi”, “hello”, “How”, “ Are”, “END”);

printf(“%s”, ptr);
```

ans:

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

char* max_len_string(int num, ...) {
    va_list args;
    va_start(args, num);

    char *max_str = va_arg(args, char*);
    int max_len = strlen(max_str);
    for (int i = 1; i < num; i++) {
        char *current_str = va_arg(args, char*);
        int current_len = strlen(current_str);

        if (current_len > max_len) {
            max_str = current_str;
            max_len = current_len;
        }
    }

    va_end(args);
    return max_str;
}

int main() {
    char *ptr = max_len_string(5, "hi", "hello", "How", " Are", "END");
    printf("%s\n", ptr);

    return 0;
}
```