Cryptography and Network Security Lab
Assignment 4

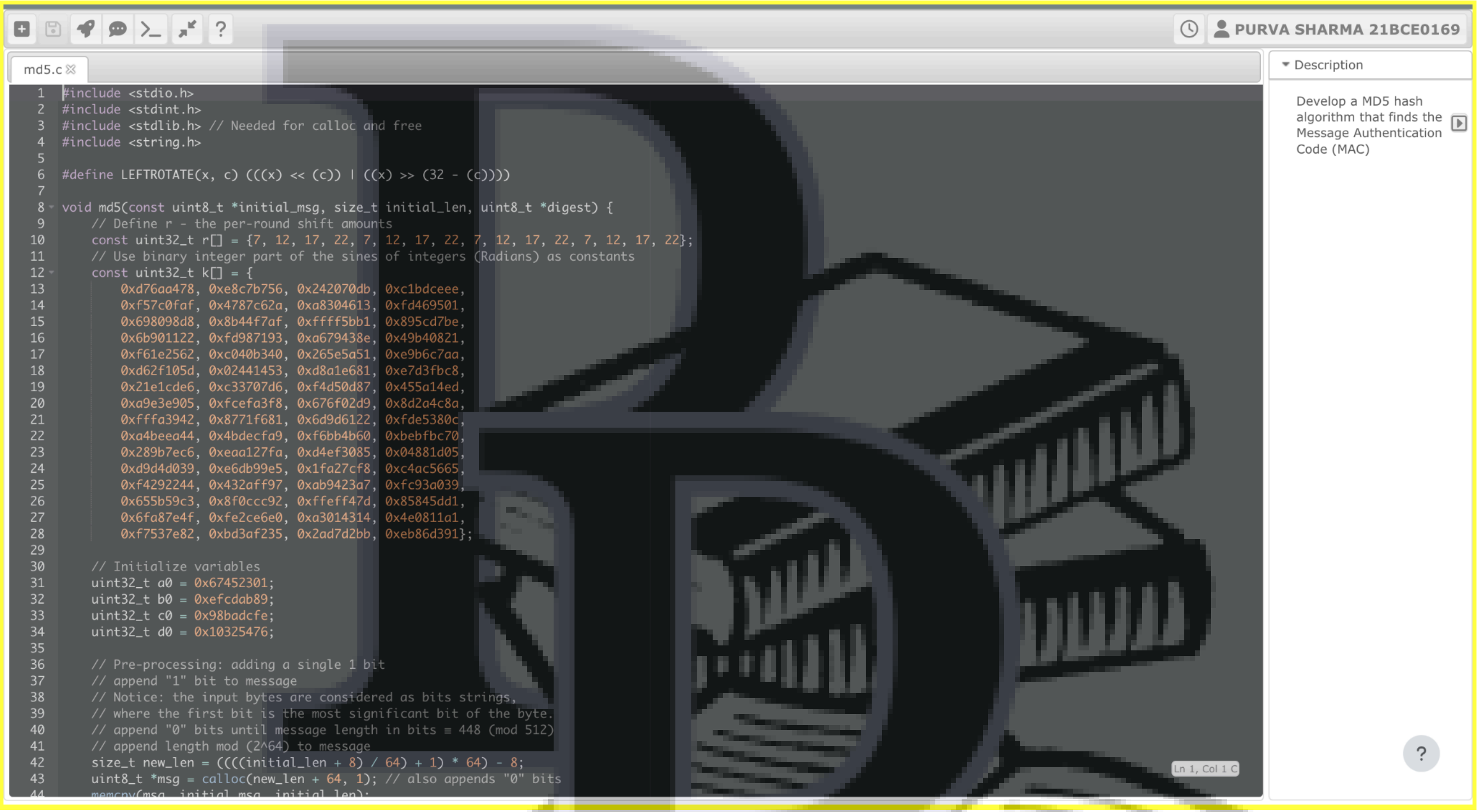
MD5

Develop a MD5 hash algorithm that finds the Message Authentication Code (MAC)

CODE:

```
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h> // Needed for calloc and free
#include <string.h>

#define LEFTROTATE(x, c) (((x) << (c)) | ((x) >> (32 - (c))))

void md5(const uint8_t *initial_msg, size_t initial_len, uint8_t *digest) {
    // Define r - the per-round shift amounts
    const uint32_t r[] = {7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22};
    // Use binary integer part of the sines of integers (Radians) as constants
    const uint32_t k[] = {
        0xd76aa478, 0xe8c7b756, 0x242070db, 0xc1bdceee,
        0xf57c0faf, 0x4787c62a, 0xa8304613, 0xfd469501,
        0x698098d8, 0x8b44f7af, 0xffff5bb1, 0x895cd7be,
        0x6b901122, 0xfd987193, 0xa679438e, 0x49b40821,
        0xf61e2562, 0xc040b340, 0x265e5a51, 0xe9b6c7aa,
        0xd62f105d, 0x02441453, 0xd8a1e681, 0xe7d3fbc8,
        0x21e1cde6, 0xc33707d6, 0xf4d50d87, 0x455a14ed,
        0xa9e3e905, 0xfcefa3f8, 0x676f02d9, 0x8d2a4c8a,
        0xfffa3942, 0x8771f681, 0x6d9d6122, 0xfde5380c,
        0xa4beea44, 0x4bdecfa9, 0xf6bb4b60, 0xbebfbc70,
        0x289b7ec6, 0xeaa127fa, 0xd4ef3085, 0x04881d05,
        0xd9d4d039, 0xe6db99e5, 0x1fa27cf8, 0xc4ac5665,
        0xf4292244, 0x432aff97, 0xab9423a7, 0xfc93a039,
        0x655b59c3, 0x8f0ccc92, 0xffeff47d, 0x85845dd1,
        0x6fa87e4f, 0xfe2ce6e0, 0xa3014314, 0x4e0811a1,
        0xf7537e82, 0xbd3af235, 0x2ad7d2bb, 0xeb86d391};

    // Initialize variables
    uint32_t a0 = 0x67452301;
    uint32_t b0 = 0xefcdab89;
    uint32_t c0 = 0x98badcfe;
    uint32_t d0 = 0x10325476;

    // Pre-processing: adding a single 1 bit
    // append "1" bit to message
    // Notice: the input bytes are considered as bits strings,
    // where the first bit is the most significant bit of the byte.
    // append "0" bits until message length in bits ≡ 448 (mod 512)
    // append length mod (2^64) to message
    size_t new_len = ((((initial_len + 8) / 64) + 1) * 64) - 8;
    uint8_t *msg = calloc(new_len + 64, 1); // also appends "0" bits
    memcpy(msg, initial_msg, initial_len);
    msg[initial_len] = 128; // write the "1" bit

    // append the length of the message (before pre-processing), in bits, as 64-bit big-endian integer
    *(uint64_t*)(msg + new_len) = initial_len * 8;
    // Process the message in successive 512-bit chunks
    // For each 512-bit chunk of message:
    for (int offset = 0; offset < new_len; offset += 64) {
        // break chunk into sixteen 32-bit words w[j], 0 ≤ j ≤ 15
        uint32_t *w = (uint32_t*)(msg + offset);
        uint32_t a = a0;
        uint32_t b = b0;
        uint32_t c = c0;
        uint32_t d = d0;

        // Main loop
        for (int i = 0; i < 64; i++) {
            uint32_t f, g;
            if (i < 16) {
                f = (b & c) | ((~b) & d);
                g = i;
            } else if (i < 32) {
                f = (d & b) | ((~d) & c);
                g = (5 * i + 1) % 16;
            } else if (i < 48) {
                f = b ^ c ^ d;
                g = (3 * i + 5) % 16;
            } else {
                f = c ^ (b | (~d));
                g = (7 * i) % 16;
            }

            // Be wary of the below definitions of a,b,c,d
            f = f + a + k[i] + w[g];
            a = d;
            d = c;
            c = b;
            b = b + LEFTROTATE(f, r[i]);
        }

        // Add this chunk's hash to result so far
        a0 += a;
        b0 += b;
        c0 += c;
        d0 += d;
    }

    // cleanup
    free(msg);

    // Put checksum in the result array
    uint32_t *p = (uint32_t*)digest;
    p[0] = a0;
    p[1] = b0;
    p[2] = c0;
    p[3] = d0;
}

int main() {
    // making a message
    const char inputstring[] = "This is a message sent by a computer user.";
    uint8_t digest[16]; // MD5 produces a 128-bit hash, so we allocate 16 bytes

    // encoding the message using the library function
    md5((const uint8_t*)inputstring, strlen(inputstring), digest);

    // printing the hash function
    printf("Hash of the input string:\n");
    for(int i = 0; i < 16; i++)
        printf("%02x", digest[i]);
    printf("\n");

    return 0;
}
```

CODE SCREENSHOT:



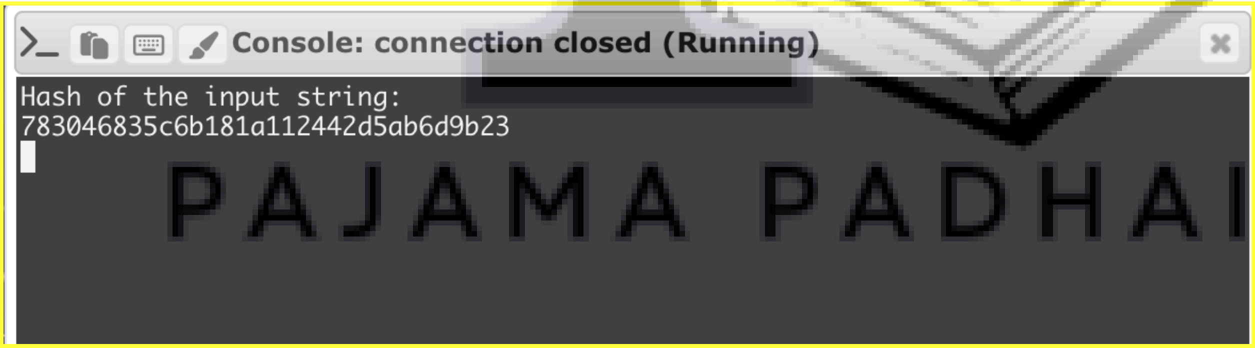
Develop a MD5 hash algorithm that finds the Message Authentication Code (MAC)

```
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h> // Needed for calloc and free
#include <string.h>

#define LEFTROTATE(x, c) (((x) << (c)) | ((x) >> (32 - (c))))

void md5(const uint8_t *initial_msg, size_t initial_len, uint8_t *digest) {
    // Define r - the per-round shift amounts
    const uint32_t r[] = {7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22};
    // Use binary integer part of the sines of integers (Radians) as constants
    const uint32_t k[] = {
        0xd76aa478, 0xe8c7b756, 0x242070db, 0xc1bdceee,
        0xf57c0faf, 0x4787c62a, 0xa8304613, 0xfd469501,
        0x698098d8, 0x8b44f7af, 0xffff5bb1, 0x895cd7be,
        0x6b901122, 0xfd987193, 0xa679438e, 0x49b40821,
        0xf61e2562, 0xc040b340, 0x265e5a51, 0xe9b6c7aa,
        0xd62f105d, 0x02441453, 0xd8a1e681, 0xe7d3fbc8,
        0x21e1cde6, 0xc33707d6, 0xf4d50d87, 0x455a14ed,
        0xa9e3e905, 0xfcefa3f8, 0x676f02d9, 0x8d2a4c8a,
        0xfffa3942, 0x8771f681, 0x6d9d6122, 0xfde5380c,
        0xa4beea44, 0x4bdecfa9, 0xf6bb4b60, 0xbebfbc70,
        0x289b7ec6, 0xeaa127fa, 0xd4ef3085, 0x04881d05,
        0xd9d4d039, 0xe6db99e5, 0x1fa27cf8, 0xc4ac5665,
        0xf4292244, 0x432aff97, 0xab9423a7, 0xfc93a039,
        0x655b59c3, 0x8f0ccc92, 0xffeff47d, 0x85845dd1,
        0x6fa87e4f, 0xfe2ce6e0, 0xa3014314, 0x4e0811a1,
        0xf7537e82, 0xbd3af235, 0x2ad7d2bb, 0xeb86d391};

    // Initialize variables
    uint32_t a0 = 0x67452301;
    uint32_t b0 = 0xefcdab89;
    uint32_t c0 = 0x98badcfe;
    uint32_t d0 = 0x10325476;

    // Pre-processing: adding a single 1 bit
    // append "1" bit to message
    // Notice: the input bytes are considered as bits strings,
    // where the first bit is the most significant bit of the byte.
    // append "0" bits until message length in bits ≡ 448 (mod 512)
    // append length mod (2^64) to message
    size_t new_len = ((((initial_len + 8) / 64) + 1) * 64) - 8;
    uint8_t *msg = calloc(new_len + 64, 1); // also appends "0" bits
    memcpy(msg, initial_msg, initial_len);
```

OUTPUT:

Hash of the input string:
783046835c6b181a112442d5ab6d9b23

OUTPUT SCREENSHOT:



```
Hash of the input string:
783046835c6b181a112442d5ab6d9b23
```

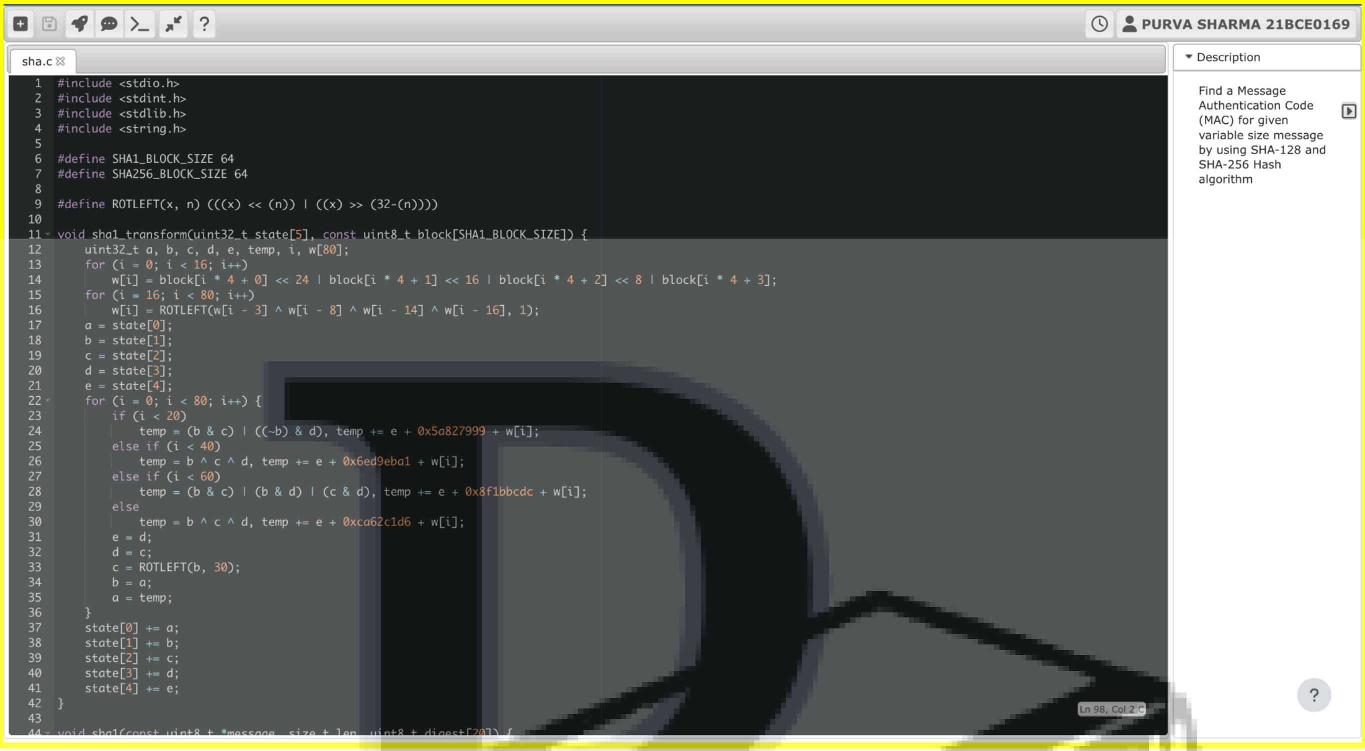
SHA

Find a Message Authentication Code (MAC) for given variable size message by using SHA-128 and SHA-256 Hash algorithm

CODE:

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

#define SHA1_BLOCK_SIZE 64
#define SHA256_BLOCK_SIZE 64

#define ROTLEFT(x, n) (((x) << (n)) | ((x) >> (32-(n))))

void sha1_transform(uint32_t state[5], const uint8_t block[SHA1_BLOCK_SIZE]) {
    uint32_t a, b, c, d, e, temp, i, w[80];
    for (i = 0; i < 16; i++)
        w[i] = block[i * 4 + 0] << 24 | block[i * 4 + 1] << 16 | block[i * 4 + 2] << 8 | block[i * 4 + 3];
    for (i = 16; i < 80; i++)
        w[i] = ROTLEFT(w[i - 3] ^ w[i - 8] ^ w[i - 14] ^ w[i - 16], 1);
    a = state[0];
    b = state[1];
    c = state[2];
    d = state[3];
    e = state[4];
    for (i = 0; i < 80; i++) {
        if (i < 20)
            temp = (b & c) | ((~b) & d), temp += e + 0x5a827999 + w[i];
        else if (i < 40)
            temp = b ^ c ^ d, temp += e + 0x6ed9eba1 + w[i];
        else if (i < 60)
            temp = (b & c) | (b & d) | (c & d), temp += e + 0x8f1bbcdc + w[i];
        else
            temp = b ^ c ^ d, temp += e + 0xca62c1d6 + w[i];
        e = d;
        d = c;
        c = ROTLEFT(b, 30);
        b = a;
        a = temp;
    }
    state[0] += a;
    state[1] += b;
    state[2] += c;
    state[3] += d;
    state[4] += e;
}

void sha1(const uint8_t *message, size_t len, uint8_t digest[20]) {
    uint32_t state[5];
    uint32_t bitlen = len * 8;
    uint8_t padding[SHA1_BLOCK_SIZE];
    int i, j;
    state[0] = 0x67452301;
    state[1] = 0xEFCDAB89;
    state[2] = 0x98BADCFE;
    state[3] = 0x10325476;
    state[4] = 0xC3D2E1F0;
    printf("Message length: %zu\n", len);
    for (i = 0; i < len / SHA1_BLOCK_SIZE; i++)
        sha1_transform(state, &message[i * SHA1_BLOCK_SIZE]);
    printf("After initial blocks transformation\n");
    memset(padding, 0, SHA1_BLOCK_SIZE);
    memcpy(padding, &message[i * SHA1_BLOCK_SIZE], len % SHA1_BLOCK_SIZE);
    padding[len % SHA1_BLOCK_SIZE] = 0x80;
    printf("After padding\n");
    if (len % SHA1_BLOCK_SIZE >= 56) {
        sha1_transform(state, padding);
        memset(padding, 0, SHA1_BLOCK_SIZE);
    }
    printf("Before final transformation\n");
    bitlen = len * 8;
    for (i = 7, j = 0; i >= 0; i--, j++)
        padding[SHA1_BLOCK_SIZE - 8 + i] = bitlen >> (j * 8);
    printf("After bit length\n");
    sha1_transform(state, padding);
    printf("After final transformation\n");
    for (i = 0; i < 5; i++) {
        digest[i * 4] = state[i] >> 24;
        digest[i * 4 + 1] = state[i] >> 16;
        digest[i * 4 + 2] = state[i] >> 8;
        digest[i * 4 + 3] = state[i];
    }
}

void print_hex(const uint8_t *data, size_t len) {
    for (size_t i = 0; i < len; ++i) {
        printf("%02x", data[i]);
    }
    printf("\n");
}

int main() {
    const char *message = "This is a test message for MAC generation.";

    uint8_t sha1_mac[20];
    sha1((const uint8_t *)message, strlen(message), sha1_mac);

    printf("SHA-1 MAC of the message:\n");
    print_hex(sha1_mac, 20);

    return 0;
}
```

CODE SCREENSHOT:



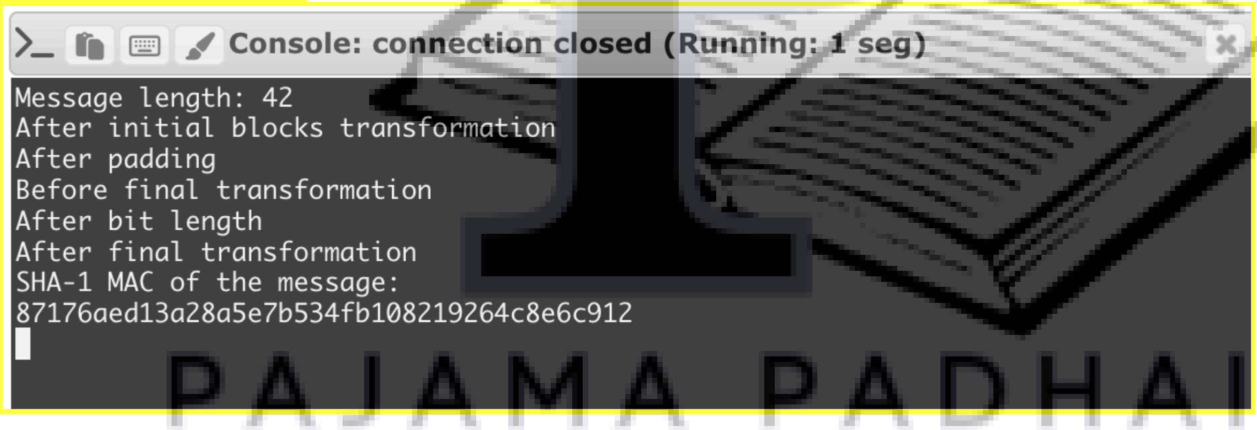
Find a Message Authentication Code (MAC) for given variable size message by using SHA-128 and SHA-256 Hash algorithm

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

#define SHA1_BLOCK_SIZE 64
#define SHA256_BLOCK_SIZE 64

#define ROTLEFT(x, n) (((x) << (n)) | ((x) >> (32-(n))))

void sha1_transform(uint32_t state[5], const uint8_t block[SHA1_BLOCK_SIZE]) {
    uint32_t a, b, c, d, e, temp, i, w[80];
    for (i = 0; i < 16; i++)
        w[i] = block[i * 4 + 0] << 24 | block[i * 4 + 1] << 16 | block[i * 4 + 2] << 8 | block[i * 4 + 3];
    for (i = 16; i < 80; i++)
        w[i] = ROTLEFT(w[i - 3] ^ w[i - 8] ^ w[i - 14] ^ w[i - 16], 1);
    a = state[0];
    b = state[1];
    c = state[2];
    d = state[3];
    e = state[4];
    for (i = 0; i < 80; i++) {
        if (i < 20)
            temp = (b & c) | ((~b) & d), temp += e + 0x5a827999 + w[i];
        else if (i < 40)
            temp = b ^ c ^ d, temp += e + 0x6ed9eba1 + w[i];
        else if (i < 60)
            temp = (b & c) | (b & d) | (c & d), temp += e + 0x8f1bbcdc + w[i];
        else
            temp = b ^ c ^ d, temp += e + 0xca62c1d6 + w[i];
        e = d;
        d = c;
        c = ROTLEFT(b, 30);
        b = a;
        a = temp;
    }
    state[0] += a;
    state[1] += b;
    state[2] += c;
    state[3] += d;
    state[4] += e;
}

void sha1(const uint8_t *message, size_t len, uint8_t digest[20]) {
```

OUTPUT:

```
Message length: 42
After initial blocks transformation
After padding
Before final transformation
After bit length
After final transformation
SHA-1 MAC of the message:
87176aed13a28a5e7b534fb108219264c8e6c912
```

OUTPUT SCREENSHOT:



```
Message length: 42
After initial blocks transformation
After padding
Before final transformation
After bit length
After final transformation
SHA-1 MAC of the message:
87176aed13a28a5e7b534fb108219264c8e6c912
```