***SAVEETHA SCHOOL OF ENGINEERING***

***SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCE***

**EXP NO 11: Calculate the message digest of the text by implementing the SHA 1 Hashing technique.**

**AIM**

To Calculate the message digest of the text by implementing the SHA 1 Hashing technique.

**PROCEDURE**

* initialize five random strings of hex characters
* The message is then [padded](https://brilliant.org/wiki/padding/) by appending a 1, followed by enough 0s until the message is 448 bits.
* begin the 80 iterations,I.
* when all the chunks have been processed, the message digest is represented as the 160-bit string comprised of the **OR** logical operator, ∨∨, of the 5 hashed values

**PROGRAM**

#include <stdio.h>

#include <stdint.h>

#include <stdlib.h>

#include <string.h>

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

void sha1(const uint8\_t \*message, size\_t len, uint32\_t hash[5]) {

uint32\_t h[] = {0x67452301, 0xEFCDAB89, 0x98BADCFE, 0x10325476, 0xC3D2E1F0};

uint64\_t bitlen = len \* 8;

uint8\_t \*padded\_message = (uint8\_t \*)malloc(len + 64); // Allocate memory for the message

memcpy(padded\_message, message, len); // Copy the original message

// Append the single '1' bit

padded\_message[len] = 0x80;

// Pad with zeros

size\_t padlen = (len < 56) ? (56 - len) : (120 - len);

memset(padded\_message + len + 1, 0, padlen);

// Append the length in bits

padded\_message[len + padlen] = (bitlen >> 56) & 0xFF;

padded\_message[len + padlen + 1] = (bitlen >> 48) & 0xFF;

padded\_message[len + padlen + 2] = (bitlen >> 40) & 0xFF;

padded\_message[len + padlen + 3] = (bitlen >> 32) & 0xFF;

padded\_message[len + padlen + 4] = (bitlen >> 24) & 0xFF;

padded\_message[len + padlen + 5] = (bitlen >> 16) & 0xFF;

padded\_message[len + padlen + 6] = (bitlen >> 8) & 0xFF;

padded\_message[len + padlen + 7] = bitlen & 0xFF;

for (size\_t i = 0; i <= len; i += 64) {

uint32\_t w[80], a = h[0], b = h[1], c = h[2], d = h[3], e = h[4];

for (size\_t j = 0; j < 16; j++)

w[j] = ((uint32\_t)padded\_message[i + j \* 4]) << 24 | ((uint32\_t)padded\_message[i + j \* 4 + 1]) << 16 |

((uint32\_t)padded\_message[i + j \* 4 + 2]) << 8 | ((uint32\_t)padded\_message[i + j \* 4 + 3]);

for (size\_t j = 16; j < 80; j++)

w[j] = LEFTROTATE(w[j - 3] ^ w[j - 8] ^ w[j - 14] ^ w[j - 16], 1);

for (size\_t j = 0; j < 80; j++) {

uint32\_t temp = LEFTROTATE(a, 5) + ((j < 20) ? ((b & c) ^ (~b & d)) + 0x5A827999

: (j < 40) ? (b ^ c ^ d) + 0x6ED9EBA1

: (j < 60) ? ((b & c) | (b & d) | (c & d)) + 0x8F1BBCDC

: (b ^ c ^ d) + 0xCA62C1D6) + e + w[j];

e = d;

d = c;

c = LEFTROTATE(b, 30);

b = a;

a = temp;

}

h[0] += a; h[1] += b; h[2] += c; h[3] += d; h[4] += e;

}

memcpy(hash, h, sizeof(uint32\_t) \* 5);

free(padded\_message); // Free the allocated memory

}

int main() {

char initial\_msg[] = "Hello, world!";

size\_t initial\_len = strlen(initial\_msg);

uint32\_t digest[5];

uint8\_t \*message = (uint8\_t \*)malloc(initial\_len); // Allocate memory for the message

memcpy(message, initial\_msg, initial\_len); // Copy the original message

sha1(message, initial\_len, digest);

printf("SHA-1 digest of '%s': ", initial\_msg);

for(int i = 0; i < 5; i++) printf("%08x", digest[i]);

printf("\n");

free(message); // Free the allocated memory

return 0;

}

**OUTPUT**

