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

#define uint32_t unsigned long
#define uint8_t unsigned char

typedef struct _MD5CONTEXT {
    uint32_t buf[4];
    uint32_t bits[2];
    uint8_t  in[64];
} MD5CONTEXT;

static void byteReverse(uint8_t *buf, size_t longs)
{
    uint32_t t;
    do {
        t = (uint32_t) ((unsigned) buf[3] << 8 | buf[2]) << 16 | ((unsigned) buf[1]
        << 8 | buf[0]);
        *(uint32_t *) buf = t;
        buf += 4;
    } while (--longs);
}

/* The four core functions - F1 is optimized somewhat */

#define F1(x, y, z) (z ^ (x & (y ^ z)))
#define F2(x, y, z) F1(z, x, y)
#define F3(x, y, z) (x ^ y ^ z)
#define F4(x, y, z) (y ^ (x | ~z))

/* This is the central step in the MD5 algorithm. */
#define MD5STEP(f, w, x, y, z, data, s) ( w += f(x, y, z) + data,  w = w<<s |
w>>(32-s),  w += x )

/*
 * The core of the MD5 algorithm, this alters an existing MD5 hash to
 * reflect the addition of 16 longwords of new data.  MD5Update blocks
 * the data and converts bytes into longwords for this routine.
 */
static void NutMD5Transform(uint32_t buf[4], uint32_t in[16])
{
    register uint32_t a, b, c, d;

    a = buf[0];
    b = buf[1];
    c = buf[2];
    d = buf[3];

    MD5STEP(F1, a, b, c, d, in[0]  + 0xd76aa478,  7);
    MD5STEP(F1, d, a, b, c, in[1]  + 0xe8c7b756, 12);
    MD5STEP(F1, c, d, a, b, in[2]  + 0x242070db, 17);
    MD5STEP(F1, b, c, d, a, in[3]  + 0xc1bdceee, 22);
    MD5STEP(F1, a, b, c, d, in[4]  + 0xf57c0faf,  7);
    MD5STEP(F1, d, a, b, c, in[5]  + 0x4787c62a, 12);
    MD5STEP(F1, c, d, a, b, in[6]  + 0xa8304613, 17);
    MD5STEP(F1, b, c, d, a, in[7]  + 0xfd469501, 22);
    MD5STEP(F1, a, b, c, d, in[8]  + 0x698098d8,  7);
    MD5STEP(F1, d, a, b, c, in[9]  + 0x8b44f7af, 12);
```

```
58        MD5STEP(F1, c, d, a, b, in[10] + 0xffff5bb1, 17);
59        MD5STEP(F1, b, c, d, a, in[11] + 0x895cd7be, 22);
60        MD5STEP(F1, a, b, c, d, in[12] + 0x6b901122,  7);
61        MD5STEP(F1, d, a, b, c, in[13] + 0xfd987193, 12);
62        MD5STEP(F1, c, d, a, b, in[14] + 0xa679438e, 17);
63        MD5STEP(F1, b, c, d, a, in[15] + 0x49b40821, 22);
64
65        MD5STEP(F2, a, b, c, d, in[1]  + 0xf61e2562,  5);
66        MD5STEP(F2, d, a, b, c, in[6]  + 0xc040b340,  9);
67        MD5STEP(F2, c, d, a, b, in[11] + 0x265e5a51, 14);
68        MD5STEP(F2, b, c, d, a, in[0]  + 0xe9b6c7aa, 20);
69        MD5STEP(F2, a, b, c, d, in[5]  + 0xd62f105d,  5);
70        MD5STEP(F2, d, a, b, c, in[10] + 0x02441453,  9);
71        MD5STEP(F2, c, d, a, b, in[15] + 0xd8a1e681, 14);
72        MD5STEP(F2, b, c, d, a, in[4]  + 0xe7d3fbc8, 20);
73        MD5STEP(F2, a, b, c, d, in[9]  + 0x21e1cde6,  5);
74        MD5STEP(F2, d, a, b, c, in[14] + 0xc33707d6,  9);
75        MD5STEP(F2, c, d, a, b, in[3]  + 0xf4d50d87, 14);
76        MD5STEP(F2, b, c, d, a, in[8]  + 0x455a14ed, 20);
77        MD5STEP(F2, a, b, c, d, in[13] + 0xa9e3e905,  5);
78        MD5STEP(F2, d, a, b, c, in[2]  + 0xfcefa3f8,  9);
79        MD5STEP(F2, c, d, a, b, in[7]  + 0x676f02d9, 14);
80        MD5STEP(F2, b, c, d, a, in[12] + 0x8d2a4c8a, 20);
81
82        MD5STEP(F3, a, b, c, d, in[5]  + 0xfffa3942,  4);
83        MD5STEP(F3, d, a, b, c, in[8]  + 0x8771f681, 11);
84        MD5STEP(F3, c, d, a, b, in[11] + 0x6d9d6122, 16);
85        MD5STEP(F3, b, c, d, a, in[14] + 0xfde5380c, 23);
86        MD5STEP(F3, a, b, c, d, in[1]  + 0xa4beea44,  4);
87        MD5STEP(F3, d, a, b, c, in[4]  + 0x4bdecfa9, 11);
88        MD5STEP(F3, c, d, a, b, in[7]  + 0xf6bb4b60, 16);
89        MD5STEP(F3, b, c, d, a, in[10] + 0xbebfbc70, 23);
90        MD5STEP(F3, a, b, c, d, in[13] + 0x289b7ec6,  4);
91        MD5STEP(F3, d, a, b, c, in[0]  + 0xeaa127fa, 11);
92        MD5STEP(F3, c, d, a, b, in[3]  + 0xd4ef3085, 16);
93        MD5STEP(F3, b, c, d, a, in[6]  + 0x04881d05, 23);
94        MD5STEP(F3, a, b, c, d, in[9]  + 0xd9d4d039,  4);
95        MD5STEP(F3, d, a, b, c, in[12] + 0xe6db99e5, 11);
96        MD5STEP(F3, c, d, a, b, in[15] + 0x1fa27cf8, 16);
97        MD5STEP(F3, b, c, d, a, in[2]  + 0xc4ac5665, 23);
98
99        MD5STEP(F4, a, b, c, d, in[0]  + 0xf4292244,  6);
100       MD5STEP(F4, d, a, b, c, in[7]  + 0x432aff97, 10);
101       MD5STEP(F4, c, d, a, b, in[14] + 0xab9423a7, 15);
102       MD5STEP(F4, b, c, d, a, in[5]  + 0xfc93a039, 21);
103       MD5STEP(F4, a, b, c, d, in[12] + 0x655b59c3,  6);
104       MD5STEP(F4, d, a, b, c, in[3]  + 0x8f0ccc92, 10);
105       MD5STEP(F4, c, d, a, b, in[10] + 0xffeff47d, 15);
106       MD5STEP(F4, b, c, d, a, in[1]  + 0x85845dd1, 21);
107       MD5STEP(F4, a, b, c, d, in[8]  + 0x6fa87e4f,  6);
108       MD5STEP(F4, d, a, b, c, in[15] + 0xfe2ce6e0, 10);
109       MD5STEP(F4, c, d, a, b, in[6]  + 0xa3014314, 15);
110       MD5STEP(F4, b, c, d, a, in[13] + 0x4e0811a1, 21);
111       MD5STEP(F4, a, b, c, d, in[4]  + 0xf7537e82,  6);
112       MD5STEP(F4, d, a, b, c, in[11] + 0xbd3af235, 10);
113       MD5STEP(F4, c, d, a, b, in[2]  + 0x2ad7d2bb, 15);
114       MD5STEP(F4, b, c, d, a, in[9]  + 0xeb86d391, 21);
115
116       buf[0] += a;
```

```
    buf[1] += b;
    buf[2] += c;
    buf[3] += d;
}

/*!
 * \brief Start MD5 accumulation.
 *
 * Start MD5 accumulation, set bit count to 0 and buffer to mysterious
 * initialization constants. Call this function to initialize every new
 * MD5 calculation.
 *
 * \param context Points to the md5 context buffer.
 */

void NutMD5Init(MD5CONTEXT *context)
{
    context->buf[0] = 0x67452301;
    context->buf[1] = 0xefcdab89;
    context->buf[2] = 0x98badcfe;
    context->buf[3] = 0x10325476;

    context->bits[0] = 0;
    context->bits[1] = 0;
}

/*!
 * \brief Update MD5 context
 *
 * Update context to reflect the concatenation of another data buffer.
 *
 * \param context Points to the md5 context buffer.
 * \param buf     Points to the data buffer
 * \param len     Length of the data buffer
 */

void NutMD5Update(MD5CONTEXT *context, uint8_t *buf, uint32_t len)
{
    uint32_t t;

    /* Update bitcount */

    t = context->bits[0];
    if ((context->bits[0] = t + ((uint32_t) len << 3)) < t) {
        context->bits[1]++;         /* Carry from low to high */
    }

    context->bits[1] += len >> 29;

    t = (t >> 3) & 0x3f;        /* Bytes already in shsInfo->data */

    /* Handle any leading odd-sized chunks */

    if (t) {
        uint8_t *p = (uint8_t *) context->in + t;

        t = 64 - t;
        if (len < t) {
            memcpy((void *)p, (void *)buf, len);
```

```
            return;
        }
        memcpy((void *)p, (void *)buf, t);
        byteReverse(context->in, 16);
        NutMD5Transform(context->buf, (uint32_t *) context->in);
        buf += t;
        len -= t;
    }

    /* Process data in 64-byte chunks */

    while (len >= 64) {
        memcpy((void *)context->in,(void *) buf, 64);
        byteReverse(context->in, 16);
        NutMD5Transform(context->buf, (uint32_t *) context->in);
        buf += 64;
        len -= 64;
    }

    /* Handle any remaining bytes of data. */

    memcpy((void *)context->in, (void *)buf, len);
}

/*!
 * \brief Final wrapup, calculate MD5 digest
 *
 * Final wrapup - pad to 64-byte boundary with the bit pattern
 * 1 0* (64-bit count of bits processed, MSB-first)
 * Fill in the digest into digest buffer
 *
 * \param context Points to the md5 context buffer.
 * \param digest  Points to the digest buffer, which must be 16 bytes long
 */

void NutMD5Final(MD5CONTEXT *context, uint8_t digest[16])
{
    unsigned int count;
    uint8_t *p;

    /* Compute number of bytes mod 64 */
    count = (context->bits[0] >> 3) & 0x3F;

    /* Set the first char of padding to 0x80.  This is safe since there is
       always at least one byte free */
    p = context->in + count;
    *p++ = 0x80;

    /* Bytes of padding needed to make 64 bytes */
    count = 64 - 1 - count;

    /* Pad out to 56 mod 64 */
    if (count < 8) {
        /* Two lots of padding:  Pad the first block to 64 bytes */
        memset(p, 0, count);
        byteReverse(context->in, 16);
        NutMD5Transform(context->buf, (uint32_t *) context->in);

        /* Now fill the next block with 56 bytes */
```

```
235         memset(context->in, 0, 56);
236     } else {
237         /* Pad block to 56 bytes */
238         memset(p, 0, count - 8);
239     }
240     byteReverse(context->in, 14);
241 
242     /* Append length in bits and transform */
243     ((uint32_t *) context->in)[14] = context->bits[0];
244     ((uint32_t *) context->in)[15] = context->bits[1];
245 
246     NutMD5Transform(context->buf, (uint32_t *) context->in);
247     byteReverse((unsigned char *) context->buf, 4);
248     memcpy(digest, context->buf, 16);
249     memset(context, 0, sizeof(MD5CONTEXT));        /* In case it's sensitive */
250 }
251 
252 
253 void main(void)
254 {
255     unsigned char md5[16];
256     MD5CONTEXT contexto;
257 
258     NutMD5Init(&contexto);
259     NutMD5Update(&contexto, "malditos terraqueos!", 20);   // o md5 tem que ser:
    712c2ff3ce5f39baf9b66635ed7cc0
260     //NutMD5Update(&contexto, "The quick brown fox jumps over the lazy dog", 43);
    // o md5 tem que ser: 9e107d9d372bb6826bd81d3542a419d6
261     NutMD5Final(&contexto, md5);
262 
263     printf("%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x \n", md5[0],md5[1],md5[2],md5[3],md5[4],
    md5[5],md5[6],md5[7],md5[8],md5[9],md5[10],md5[11],md5[12],md5[13],md5[14],md5[15
    ]);
264     while(1) {}
265 
266 }
267 
```