MD5 and SHA-1 – Pseudo code

Taken from Wikipedia

MD5

```
//Note: All variables are unsigned 32 bit and wrap modulo 2^32 when
calculating
var int[64] s, K

//s specifies the per-round shift amounts
s[ 0..15] := { 7, 12, 17, 22,  7, 12, 17, 22,  7, 12, 17, 22,  7, 12,
17, 22}
s[16..31] := { 5,  9, 14, 20,  5,  9, 14, 20,  5,  9, 14, 20,  5,  9,
14, 20}
s[32..47] := { 4, 11, 16, 23,  4, 11, 16, 23,  4, 11, 16, 23,  4, 11,
16, 23}
s[48..63] := { 6, 10, 15, 21,  6, 10, 15, 21,  6, 10, 15, 21,  6, 10,
15, 21}

//Use binary integer part of the sines of integers (Radians) as
constants:
for i from 0 to 63
    K[i] := floor(abs(sin(i + 1)) × (2 pow 32))
end for
//(Or just use the following table):
K[ 0.. 3] := { 0xd76aa478, 0xe8c7b756, 0x242070db, 0xc1bdceee }
K[ 4.. 7] := { 0xf57c0faf, 0x4787c62a, 0xa8304613, 0xfd469501 }
K[ 8..11] := { 0x698098d8, 0x8b44f7af, 0xffff5bb1, 0x895cd7be }
K[12..15] := { 0x6b901122, 0xfd987193, 0xa679438e, 0x49b40821 }
K[16..19] := { 0xf61e2562, 0xc040b340, 0x265e5a51, 0xe9b6c7aa }
K[20..23] := { 0xd62f105d, 0x02441453, 0xd8a1e681, 0xe7d3fbc8 }
K[24..27] := { 0x21e1cde6, 0xc33707d6, 0xf4d50d87, 0x455a14ed }
K[28..31] := { 0xa9e3e905, 0xfcefa3f8, 0x676f02d9, 0x8d2a4c8a }
K[32..35] := { 0xfffa3942, 0x8771f681, 0x6d9d6122, 0xfde5380c }
K[36..39] := { 0xa4beea44, 0x4bdecfa9, 0xf6bb4b60, 0xbebfbc70 }
K[40..43] := { 0x289b7ec6, 0xeaa127fa, 0xd4ef3085, 0x04881d05 }
K[44..47] := { 0xd9d4d039, 0xe6db99e5, 0x1fa27cf8, 0xc4ac5665 }
K[48..51] := { 0xf4292244, 0x432aff97, 0xab9423a7, 0xfc93a039 }
K[52..55] := { 0x655b59c3, 0x8f0ccc92, 0xffeff47d, 0x85845dd1 }
K[56..59] := { 0x6fa87e4f, 0xfe2ce6e0, 0xa3014314, 0x4e0811a1 }
K[60..63] := { 0xf7537e82, 0xbd3af235, 0x2ad7d2bb, 0xeb86d391 }

//Initialize variables:
var int a0 := 0x67452301   //A
var int b0 := 0xefcdab89   //B
var int c0 := 0x98badcfe   //C
var int d0 := 0x10325476   //D

//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.[37]


//Pre-processing: padding with zeros
append "0" bit until message length in bit ≡ 448 (mod 512)
append length mod (2 pow 64) to message


//Process the message in successive 512-bit chunks:
for each 512-bit chunk of message
    break chunk into sixteen 32-bit words M[j], 0 ≤ j ≤ 15
//Initialize hash value for this chunk:
    var int A := a0
    var int B := b0
    var int C := c0
    var int D := d0
//Main loop:
    for i from 0 to 63
        if 0 ≤ i ≤ 15 then
            F := (B and C) or ((not B) and D)
            g := i
        else if 16 ≤ i ≤ 31
            F := (D and B) or ((not D) and C)
            g := (5×i + 1) mod 16
        else if 32 ≤ i ≤ 47
            F := B xor C xor D
            g := (3×i + 5) mod 16
        else if 48 ≤ i ≤ 63
            F := C xor (B or (not D))
            g := (7×i) mod 16
        dTemp := D
        D := C
        C := B
        B := B + leftrotate((A + F + K[i] + M[g]), s[i])
        A := dTemp
    end for
//Add this chunk's hash to result so far:
    a0 := a0 + A
    b0 := b0 + B
    c0 := c0 + C
    d0 := d0 + D
end for

var char digest[16] := a0 append b0 append c0 append d0 //(Output is
in little-endian)

//leftrotate function definition
leftrotate (x, c)
    return (x << c) binary or (x >> (32-c));
```

SHA-1

```
Note: All variables are unsigned 32 bits and wrap modulo 2^32 when
calculating

Initialize variables:
h0 := 0x67452301
h1 := 0xEFCDAB89
h2 := 0x98BADCFE
h3 := 0x10325476
h4 := 0xC3D2E1F0

Pre-processing:
append a single "1" bit to message
append "0" bits until message length ≡ 448 ≡ -64 (mod 512)
append length of message (before pre-processing), in bits as 64-bit big-
endian integer to message

Process the message in successive 512-bit chunks:
break message into 512-bit chunks
for each chunk
    break chunk into sixteen 32-bit big-endian words w(i), 0 ≤ i ≤ 15

    Extend the sixteen 32-bit words into eighty 32-bit words:
    for i from 16 to 79
       w(i) := (w(i-3) xor w(i-8) xor w(i-14) xor w(i-16)) leftrotate 1

    Initialize hash value for this chunk:
    a := h0
    b := h1
    c := h2
    d := h3
    e := h4

    Main loop:
    for i from 0 to 79
        if 0 ≤ i ≤ 19 then
            f := (b and c) or ((not b) and d)
            k := 0x5A827999
        else if 20 ≤ i ≤ 39
            f := b xor c xor d
            k := 0x6ED9EBA1
        else if 40 ≤ i ≤ 59
            f := (b and c) or (b and d) or (c and d)
            k := 0x8F1BBCDC
        else if 60 ≤ i ≤ 79
            f := b xor c xor d
            k := 0xCA62C1D6

        temp := (a leftrotate 5) + f + e + k + w(i)
        e := d
        d := c
        c := b leftrotate 30
        b := a
        a := temp

    Add this chunk's hash to result so far:
    h0 := h0 + a
    h1 := h1 + b
    h2 := h2 + c
    h3 := h3 + d
    h4 := h4 + e

digest = hash = h0 append h1 append h2 append h3 append h4 (expressed as
big-endian)
```