#include <string>

#include <iostream>

#include <cstring>

#include <sstream>

#include <iomanip>

#include <iostream>

#include <array>

using namespace std;

/\*

Массив констант.

Каждое значение (0—63) представляет собой первые 32 бита дробных частей кубических корней первых 64 простых чисел (2—311).

Используется на 3 шаге алгоритма.

\*/

constexpr array<uint32\_t, 64> K = {

0x428a2f98, 0x71374491, 0xb5c0fbcf, 0xe9b5dba5,

0x3956c25b, 0x59f111f1, 0x923f82a4, 0xab1c5ed5,

0xd807aa98, 0x12835b01, 0x243185be, 0x550c7dc3,

0x72be5d74, 0x80deb1fe, 0x9bdc06a7, 0xc19bf174,

0xe49b69c1, 0xefbe4786, 0x0fc19dc6, 0x240ca1cc,

0x2de92c6f, 0x4a7484aa, 0x5cb0a9dc, 0x76f988da,

0x983e5152, 0xa831c66d, 0xb00327c8, 0xbf597fc7,

0xc6e00bf3, 0xd5a79147, 0x06ca6351, 0x14292967,

0x27b70a85, 0x2e1b2138, 0x4d2c6dfc, 0x53380d13,

0x650a7354, 0x766a0abb, 0x81c2c92e, 0x92722c85,

0xa2bfe8a1, 0xa81a664b, 0xc24b8b70, 0xc76c51a3,

0xd192e819, 0xd6990624, 0xf40e3585, 0x106aa070,

0x19a4c116, 0x1e376c08, 0x2748774c, 0x34b0bcb5,

0x391c0cb3, 0x4ed8aa4a, 0x5b9cca4f, 0x682e6ff3,

0x748f82ee, 0x78a5636f, 0x84c87814, 0x8cc70208,

0x90befffa, 0xa4506ceb, 0xbef9a3f7, 0xc67178f2

};

uint8\_t m\_data[64]; /// Тут мы будем храник хэш. данные

uint64\_t m\_bitlen(0);/// Длина битов

uint32\_t m\_blocklen(0);/// Кол-во блоков

/\* Массив из квадратов первых 8 простых чисел \*/

uint32\_t m\_state[8] = {

0x6a09e667,

0xbb67ae85,

0x3c6ef372,

0xa54ff53a,

0x510e527f,

0x9b05688c,

0x1f83d9ab,

0x5be0cd19

};

void reset()

{

uint8\_t nm\_data[64];

\*m\_data = \*nm\_data;

uint64\_t nm\_bitlen(0);

m\_bitlen = nm\_bitlen;

uint32\_t nm\_blocklen(0);

m\_blocklen = nm\_blocklen;

uint32\_t nm\_state[8] = {

0x6a09e667,

0xbb67ae85,

0x3c6ef372,

0xa54ff53a,

0x510e527f,

0x9b05688c,

0x1f83d9ab,

0x5be0cd19

};

memcpy(nm\_state, m\_state, 8);

}

uint32\_t rotr(uint32\_t x, uint32\_t n)

{

return (x >> n) | (x << (32 - n));

}

uint32\_t choose(uint32\_t e, uint32\_t f, uint32\_t g)

{

return (e & f) ^ (~e & g);

}

uint32\_t majority(uint32\_t a, uint32\_t b, uint32\_t c)

{

return (a & (b | c)) | (b & c);

}

uint32\_t sig0(uint32\_t x)

{

return rotr(x, 7) ^ rotr(x, 18) ^ (x >> 3);

}

uint32\_t sig1(uint32\_t x)

{

return rotr(x, 17) ^ rotr(x, 19) ^ (x >> 10);

}

/// Здесь происходит само преобразование в хэш

void transform()

{

uint32\_t maj, xorA, ch, xorE, sum, newA, newE, m[64];

uint32\_t state[8];

///

for (uint8\_t i = 0, j = 0; i < 16; i++, j += 4)

{

m[i] = (m\_data[j] << 24) | (m\_data[j + 1] << 16) | (m\_data[j + 2] << 8) | (m\_data[j + 3]);

}

for (uint8\_t k = 16; k < 64; k++)

{

m[k] = sig1(m[k - 2]) + m[k - 7] + sig0(m[k - 15]) + m[k - 16];

}

for (uint8\_t i = 0; i < 8; i++)

{

state[i] = m\_state[i];

}

/// Тут сам процесс шифрования (сдвиги по алгоритму SHA-256)

for (uint8\_t i = 0; i < 64; i++)

{

maj = majority(state[0], state[1], state[2]);

xorA = rotr(state[0], 2) ^ rotr(state[0], 13) ^ rotr(state[0], 22);

ch = choose(state[4], state[5], state[6]);

xorE = rotr(state[4], 6) ^ rotr(state[4], 11) ^ rotr(state[4], 25);

sum = m[i] + K[i] + state[7] + ch + xorE;

newA = xorA + maj + sum;

newE = state[3] + sum;

state[7] = state[6];

state[6] = state[5];

state[5] = state[4];

state[4] = newE;

state[3] = state[2];

state[2] = state[1];

state[1] = state[0];

state[0] = newA;

}

for (uint8\_t i = 0; i < 8; i++)

{

m\_state[i] += state[i];

}

}

void pad()

{

uint64\_t i = m\_blocklen;

uint8\_t end = m\_blocklen < 56 ? 56 : 64;

m\_data[i++] = 0x80;/// Добавили 1 в конец

while (i < end)

{

m\_data[i++] = 0x00;/// Добавили нули

}

if (m\_blocklen >= 56)

{

transform();

memset(m\_data, 0, 56);

}

/// По формуле порядка байтов

m\_bitlen += m\_blocklen \* 8;

m\_data[63] = m\_bitlen;

m\_data[62] = m\_bitlen >> 8;

m\_data[61] = m\_bitlen >> 16;

m\_data[60] = m\_bitlen >> 24;

m\_data[59] = m\_bitlen >> 32;

m\_data[58] = m\_bitlen >> 40;

m\_data[57] = m\_bitlen >> 48;

m\_data[56] = m\_bitlen >> 56;

transform();

}

void revert(uint8\_t\* hash)

{

for (uint8\_t i = 0; i < 4; i++)

{

for (uint8\_t j = 0; j < 8; j++)

{

hash[i + (j \* 4)] = (m\_state[j] >> (24 - i \* 8)) & 0x000000ff;

}

}

}

string toString(const uint8\_t\* digest)

{

stringstream s;

s << setfill('0') << hex;

for (uint8\_t i = 0; i < 32; i++)

{

s << setw(2) << (unsigned int)digest[i];

}

return s.str();

}

void update(const uint8\_t\* data, size\_t length) /// ф-я принимает символы текста и его длину

{

for (size\_t i = 0; i < length; i++)

{

m\_data[m\_blocklen++] = data[i]; /// Делим на блоки на 512 бит

if (m\_blocklen == 64)

{

transform(); /// мы его запустим если кол-во блоков станет равным 64

m\_bitlen += 512;

m\_blocklen = 0;

}

}

}

void update(const std::string& data) /// Если мы передали строку, то преобразовываем в uint8\_t

{

update(reinterpret\_cast<const uint8\_t\*>(data.c\_str()), data.size());/// преобразовываем текст в битовое представление по алгоритму

} ///

uint8\_t\* digest()

{

uint8\_t\* hash = new uint8\_t[32]; /// мы создали указатель, который указывает на коллекцию этих блоков

pad();

revert(hash);

return hash;

}

int main()

{

string word = "Maria";/// Получаем строку

update(word); /// Отправили строку в ф-ю update

cout << toString(digest()) << endl;

}