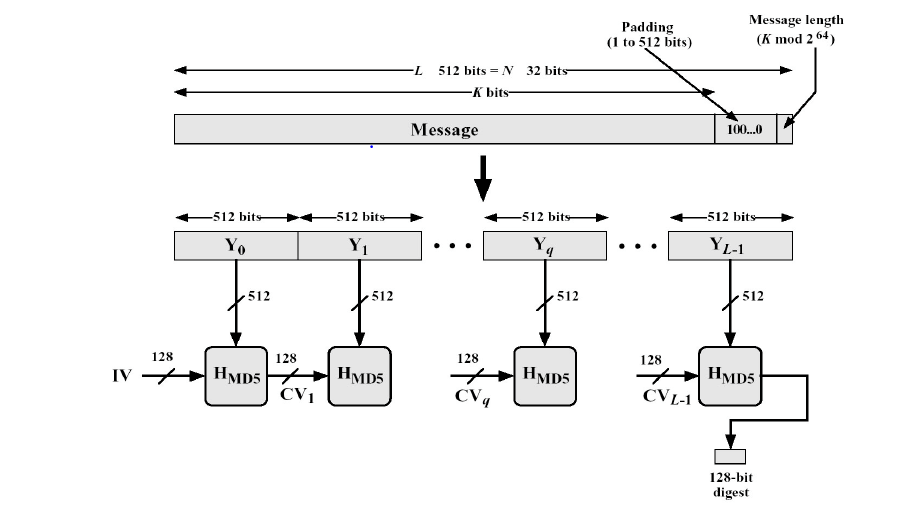
**MD5算法程序设计与实现**

**算法原理概述：**

* MD5 即Message-Digest Algorithm 5 (信息-摘要算法5)
  + MD4 (1990)、MD5(1992, RFC 1321)由Ron Rivest发明，是广泛使用的Hash 算法，用于确保信息传输的完整性和一致性。
  + MD5 使用little-endian(小端模式)，输入任意不定长度信息，以512-bit进行分组，生成四个32-bit 数据，最后联合输出固定 128-bit 的信息摘要。
  + MD5 算法的基本过程为：填充、分块、缓冲区初始化、循环压 缩、得出结果。
  + MD5 不是足够安全的。
    - Hans Dobbertin在1996年找到了两个不同的512-bit 块,它们 在MD5 计算下产生相同的hash 值。
    - 至今还没有真正找到两个不同的消息，它们的MD5 的hash 值相等。

基本的流程图如下：



算法逻辑：

* 填充padding
  + 在长度为Kbits 的原始消息数据尾部填充长度为Pbits的标识 100…0，1< P < 512 (即至少要填充1个bit)，使得填充后的消息位数为：K + P ≡ 448 (mod 512).
    - 注意到当K ≡ 448 (mod 512) 时，需要P= 512.
  + 再向上述填充好的消息尾部附加K 值的低64位(即K mod 264)， 最后得到一个长度位数为K + P+ 64= 0 (mod 512) 的消息。
* 分块
  + 把填充后的消息结果分割为L 个512-bit 分组：Y0, Y1, …, YL-1。
  + 分组结果也可表示成N 个32-bit 字M0, M1, …, MN-1，N = L\*16。
* 初始化
  + 初始化一个128-bit 的MD 缓冲区，记为CVq，表示成4个32-bit寄存器(A, B, C, D)；CV0 = IV。迭代在MD 缓冲区进行，最后一步的128-bit 输出即为算法结果。

**总体结构：**

class MD5 {

public:

    MD5(); //构造函数初始化

    void Update(unsigned char\*, unsigned int); //对数据进行分组，并进行加密

    void Padding(unsigned char[16]); //数据填充

    void Transform(unsigned int[4], unsigned char[64]); //轮换操作函数

    void Encode(unsigned char\*, unsigned int\*, unsigned int); //加密信息变为字符

    void Decode(unsigned int\*, unsigned char\*, unsigned int); //把字符变为数字

private:

    unsigned int count[2]; //记录数据位数

    unsigned int regist[4]; //记录用于保存对512bits信息加密的中间结果或者最终结果

    unsigned char buffer[64]; //512位缓冲区

    unsigned char PADDING[64] = { 0x80,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 }; //用于数据填充

};

md5.h：定义了一些宏定义以及一个md5类.

md5.cpp：实现md5类的方法.

main.cpp：测试实现的md5算法的有效性。

**数据结构：**

用到了类和数组，定义在上面已经给出。

除此之外，我直接使用了RCF 1321中的宏定义，这样既可以提高程序运行的效率，也可以提高写代码的效率：

#define S11 7

#define S12 12

#define S13 17

#define S14 22

#define S21 5

#define S22 9

#define S23 14

#define S24 20

#define S31 4

#define S32 11

#define S33 16

#define S34 23

#define S41 6

#define S42 10

#define S43 15

#define S44 21

//函数F是按逐位方式操作：如果X，那么Y，否则Z

//函数H是逐位奇偶操作符

#define F(x, y, z) (((x) & (y)) | ((~x) & (z)))

#define G(x, y, z) (((x) & (z)) | ((y) & (~z)))

#define H(x, y, z) ((x) ^ (y) ^ (z))

#define I(x, y, z) ((y) ^ ((x) | (~z)))

using namespace std;

//向右环移n个单位

#define ROTATELEFT(num, n) (((num) << (n)) | ((num) >> (32 - (n))))

#define FF(a, b, c, d, x, s, ac) \

{ \

(a) += F((b), (c), (d)) + (x) + ac; \

(a) = ROTATELEFT((a), (s)); \

(a) += (b); \

}

#define GG(a, b, c, d, x, s, ac) \

{ \

(a) += G((b), (c), (d)) + (x) + ac; \

(a) = ROTATELEFT((a), (s)); \

(a) += (b); \

}

#define HH(a, b, c, d, x, s, ac) \

{ \

(a) += H((b), (c), (d)) + (x) + ac; \

(a) = ROTATELEFT((a), (s)); \

(a) += (b); \

}

#define II(a, b, c, d, x, s, ac) \

{ \

(a) += I((b), (c), (d)) + (x) + ac; \

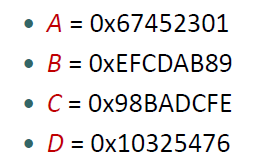
(a) = ROTATELEFT((a), (s)); \

(a) += (b); \

}

**模块分解：**

首先初始化数据位数和寄存器，寄存器163进制初值在ppt上由标明：



//初始化

MD5::MD5() {

    count[0] = 0;

    count[1] = 0;

    regist[0] = 0x67452301;

    regist[1] = 0xEFCDAB89;

    regist[2] = 0x98BADCFE;

    regist[3] = 0x10325476;

}

填充padding：要实现的就是数据补足，主要就是填充数据是的其满足其位长对512求余的结果等于448（56位），所以如果所以当其所剩余的数小于56字节，则填充56-index字节，否则填充120-index字节。填充时，在后面填充一个1和无数个0，直到满足上面的条件时才停止用0对数据的填充：

    unsigned int index = 0, len = 0;

    unsigned char bits[8];

    index = (count[0] / 8) % 64;

    len = (index < 56) ? (56 - index) : (120 - index);

    Encode(bits, count, 8);

    //在信息的后面填充一个1和无数个0

    Update(PADDING, len);

    Update(bits, 8);

    Encode(digest, regist, 16);

Update：这个模块实现将消息尾部附加K 值的低64位，同时解决unsigned int无法储存大数据的问题，通过以下操作，可以将两个16位的数连在一起，生成一个32的二进制串：

    count[0] += len \* 8;

    if (count[0] < (len \* 8)) {

        count[1]++;

    }

总代码如下：

void MD5::Update(unsigned char\* input, unsigned int len) {

    unsigned int i = 0, index = 0, templen = 0;

    //当前状态的位数对64取余

    index = (count[0] / 8) % 64;

    templen = 64 - index;

    //解决unsignde int无法储存极大数据导致溢出的问题

    count[0] += len \* 8;

    if (count[0] < (len \* 8)) {

        count[1]++;

    }

    count[1] += len >> 29;

    //当其输入字节数大于其可以补足64字节的字节数，进行补足

    if (len >= templen) {

        memcpy(&buffer[index], input, templen);

        Transform(regist, buffer);

        for (i = templen; i + 64 <= len; i += 64) {

            Transform(regist, &input[i]);

        }

        index = 0;

    }

    else {

        i = 0;

    }

    //放置剩余数据

    memcpy(&buffer[index], &input[i], len - i);

}

Encode：实现在填充的数据后附加一个以64位二进制表示的填充前数据长度，最后得到相应的信息。通过把数字的8、8-16、16-24、24-32分别赋值给字符实现：

void MD5::Encode(unsigned char\* output, unsigned int\* input, unsigned int len) {

    unsigned int i = 0, j = 0;

    while (j < len) {

        //把数字的8、8-16、16-24、24-32分别赋值给字符

        output[j] = input[i] & 0xFF;

        output[j + 1] = (input[i] >> 8) & 0xFF;

        output[j + 2] = (input[i] >> 16) & 0xFF;

        output[j + 3] = (input[i] >> 24) & 0xFF;

        i++;

        j += 4;

    }

}

Decode：解码操作，将字符串转换为数字，便于轮换：

void MD5::Decode(unsigned int\* output, unsigned char\* input, unsigned int len) {

    unsigned int i = 0, j = 0;

    while (j < len) {

        //利用位操作，把四个单位为1字节的字符，合成一个单位为4字节的数字

        output[i] = (input[j]) |

            (input[j + 1] << 8) |

            (input[j + 2] << 16) |

            (input[j + 3] << 24);

        i++;

        j += 4;

    }

}

Transform：对512位的block数据进行加密，对512位数据进行处理，每次处理使用之前的宏定义完成四轮轮转运算，随后在末尾分别再加上原始数据：

void MD5::Transform(unsigned int regist\_[4], unsigned char block[64]) {

    unsigned int a = regist\_[0];

    unsigned int b = regist\_[1];

    unsigned int c = regist\_[2];

    unsigned int d = regist\_[3];

    unsigned int x[16];

    Decode(x, block, 64);

    // Round 1

    FF(a, b, c, d, x[0], S11, 0xd76aa478);

    FF(d, a, b, c, x[1], S12, 0xe8c7b756);

    FF(c, d, a, b, x[2], S13, 0x242070db);

    FF(b, c, d, a, x[3], S14, 0xc1bdceee);

    FF(a, b, c, d, x[4], S11, 0xf57c0faf);

    FF(d, a, b, c, x[5], S12, 0x4787c62a);

    FF(c, d, a, b, x[6], S13, 0xa8304613);

    FF(b, c, d, a, x[7], S14, 0xfd469501);

    FF(a, b, c, d, x[8], S11, 0x698098d8);

    FF(d, a, b, c, x[9], S12, 0x8b44f7af);

    FF(c, d, a, b, x[10], S13, 0xffff5bb1);

    FF(b, c, d, a, x[11], S14, 0x895cd7be);

    FF(a, b, c, d, x[12], S11, 0x6b901122);

    FF(d, a, b, c, x[13], S12, 0xfd987193);

    FF(c, d, a, b, x[14], S13, 0xa679438e);

    FF(b, c, d, a, x[15], S14, 0x49b40821);

    // Round 2

    GG(a, b, c, d, x[1], S21, 0xf61e2562);

    GG(d, a, b, c, x[6], S22, 0xc040b340);

    GG(c, d, a, b, x[11], S23, 0x265e5a51);

    GG(b, c, d, a, x[0], S24, 0xe9b6c7aa);

    GG(a, b, c, d, x[5], S21, 0xd62f105d);

    GG(d, a, b, c, x[10], S22, 0x2441453);

    GG(c, d, a, b, x[15], S23, 0xd8a1e681);

    GG(b, c, d, a, x[4], S24, 0xe7d3fbc8);

    GG(a, b, c, d, x[9], S21, 0x21e1cde6);

    GG(d, a, b, c, x[14], S22, 0xc33707d6);

    GG(c, d, a, b, x[3], S23, 0xf4d50d87);

    GG(b, c, d, a, x[8], S24, 0x455a14ed);

    GG(a, b, c, d, x[13], S21, 0xa9e3e905);

    GG(d, a, b, c, x[2], S22, 0xfcefa3f8);

    GG(c, d, a, b, x[7], S23, 0x676f02d9);

    GG(b, c, d, a, x[12], S24, 0x8d2a4c8a);

    // Round 3

    HH(a, b, c, d, x[5], S31, 0xfffa3942);

    HH(d, a, b, c, x[8], S32, 0x8771f681);

    HH(c, d, a, b, x[11], S33, 0x6d9d6122);

    HH(b, c, d, a, x[14], S34, 0xfde5380c);

    HH(a, b, c, d, x[1], S31, 0xa4beea44);

    HH(d, a, b, c, x[4], S32, 0x4bdecfa9);

    HH(c, d, a, b, x[7], S33, 0xf6bb4b60);

    HH(b, c, d, a, x[10], S34, 0xbebfbc70);

    HH(a, b, c, d, x[13], S31, 0x289b7ec6);

    HH(d, a, b, c, x[0], S32, 0xeaa127fa);

    HH(c, d, a, b, x[3], S33, 0xd4ef3085);

    HH(b, c, d, a, x[6], S34, 0x4881d05);

    HH(a, b, c, d, x[9], S31, 0xd9d4d039);

    HH(d, a, b, c, x[12], S32, 0xe6db99e5);

    HH(c, d, a, b, x[15], S33, 0x1fa27cf8);

    HH(b, c, d, a, x[2], S34, 0xc4ac5665);

    // Round 4

    II(a, b, c, d, x[0], S41, 0xf4292244);

    II(d, a, b, c, x[7], S42, 0x432aff97);

    II(c, d, a, b, x[14], S43, 0xab9423a7);

    II(b, c, d, a, x[5], S44, 0xfc93a039);

    II(a, b, c, d, x[12], S41, 0x655b59c3);

    II(d, a, b, c, x[3], S42, 0x8f0ccc92);

    II(c, d, a, b, x[10], S43, 0xffeff47d);

    II(b, c, d, a, x[1], S44, 0x85845dd1);

    II(a, b, c, d, x[8], S41, 0x6fa87e4f);

    II(d, a, b, c, x[15], S42, 0xfe2ce6e0);

    II(c, d, a, b, x[6], S43, 0xa3014314);

    II(b, c, d, a, x[13], S44, 0x4e0811a1);

    II(a, b, c, d, x[4], S41, 0xf7537e82);

    II(d, a, b, c, x[11], S42, 0xbd3af235);

    II(c, d, a, b, x[2], S43, 0x2ad7d2bb);

    II(b, c, d, a, x[9], S44, 0xeb86d391);

    regist\_[0] += a;

    regist\_[1] += b;

    regist\_[2] += c;

    regist\_[3] += d;

}

最后时main.cpp测试函数，用来测试md5算法的正确性：

#include "md5.h"

int main() {

    MD5 md5;

    int i;

    unsigned char input[1000];

    cin >> input;

    unsigned char output[16];

    md5.Update(input, strlen((char \*)input));

    md5.Padding(output);

    cout << "加密前:" << input << "\n加密后:";

    for (i = 0; i < 16; i++) {

        printf("%02x", output[i]);

    }

    cout << endl;

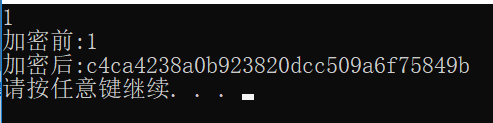
    system("pause");

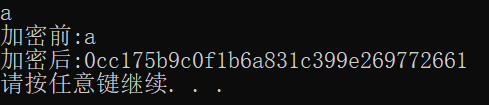
    return 0;

}

**编译运行结果：**

程序运行的结果如下：





在在线md5加密网站中验证结果：





这说明md5加密算法已经成功实现。