БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Кафедра информатики

Факультет ИНО

Специальность ИиТП

Контрольная работа № 1

по дисциплине «Методы защиты информации»

Выполнил студент: Дегтярев А.А.

группа 393551

Зачетная книжка № 902021-26

Минск 2018

**КР №1**

#### MD5

Создать и протестировать алгоритм формирования Хэш-функции MD5 на языке высокого уровня.

Порядок использования программы:



Необходимо указать файл для чтения с помощью флага **–i**

**Использовать флаг MD5(md5) для указания алгоритма**

Алгоритм читает файл блоками по 512, как только читается последний блок, выполняется его расширения до 448 и далее указывается размер целого файла.   
  
Для использования алгоритма в коде представлено API

**void** md5\_process\_block(uint32\_t\* block, uint32\_t\* h0,uint32\_t\* h1,uint32\_t\* h2,uint32\_t\* h3);

MD5 – алгоритм формирования хэша фиксированного размера в 128бит.

Для генерации представлен метод, с аргументами(*указатель на блок данных по 512бит*, *указатель на части hash функции сформированной на предыдущем раунде*)

Содержимое тестового файла:



Пример использования:



Результат:

7767f2d709da925be0f77b91f2ec5edb

Для проверки правильности, воспользовался встроенной UNIX утилитой md5 и получил такой же результат:  
  


Исходные файлы:  
MD5.h

#ifndef MZI\_MD5\_H  
#define MZI\_MD5\_H  
  
#include <cstdint>  
  
#define LEFTROTATE(x, c) (((x) << (c)) | ((x) >> (32 - (c))))  
  
**void** md5\_process\_block(uint32\_t\* block, uint32\_t\* h0,uint32\_t\* h1,uint32\_t\* h2,uint32\_t\* h3);  
  
**const** uint32\_t md5\_round\_shifts[64] = {  
 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22,  
 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20,  
 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23,  
 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21  
};  
  
//binary integer part of the sines of integers  
**const** uint32\_t md5\_k[64] = {  
 0xd76aa478, 0xe8c7b756, 0x242070db, 0xc1bdceee,  
 0xf57c0faf, 0x4787c62a, 0xa8304613, 0xfd469501,  
 0x698098d8, 0x8b44f7af, 0xffff5bb1, 0x895cd7be,  
 0x6b901122, 0xfd987193, 0xa679438e, 0x49b40821,  
 0xf61e2562, 0xc040b340, 0x265e5a51, 0xe9b6c7aa,  
 0xd62f105d, 0x02441453, 0xd8a1e681, 0xe7d3fbc8,  
 0x21e1cde6, 0xc33707d6, 0xf4d50d87, 0x455a14ed,  
 0xa9e3e905, 0xfcefa3f8, 0x676f02d9, 0x8d2a4c8a,  
 0xfffa3942, 0x8771f681, 0x6d9d6122, 0xfde5380c,  
 0xa4beea44, 0x4bdecfa9, 0xf6bb4b60, 0xbebfbc70,  
 0x289b7ec6, 0xeaa127fa, 0xd4ef3085, 0x04881d05,  
 0xd9d4d039, 0xe6db99e5, 0x1fa27cf8, 0xc4ac5665,  
 0xf4292244, 0x432aff97, 0xab9423a7, 0xfc93a039,  
 0x655b59c3, 0x8f0ccc92, 0xffeff47d, 0x85845dd1,  
 0x6fa87e4f, 0xfe2ce6e0, 0xa3014314, 0x4e0811a1,  
 0xf7537e82, 0xbd3af235, 0x2ad7d2bb, 0xeb86d391  
};  
  
**const int** md5\_a0 = 0x67452301;  
**const int** md5\_b0 = 0xefcdab89;  
**const int** md5\_c0 = 0x98badcfe;  
**const int** md5\_d0 = 0x10325476;  
  
#endif //MZI\_MD5\_H

MD5.cpp

#include "md5.h"  
**void** md5\_process\_block(uint32\_t\* block, uint32\_t\* h0,uint32\_t\* h1,uint32\_t\* h2,uint32\_t\* h3 ){  
 uint32\_t a = \*h0;  
 uint32\_t b = \*h1;  
 uint32\_t c = \*h2;  
 uint32\_t d = \*h3;  
  
 uint32\_t i;  
 **for**(i = 0; i<64; i++) {  
 uint32\_t f, g;  
  
 **if** (i < 16) {  
 f = (b & c) | ((~b) & d);  
 g = i;  
 } **else if** (i < 32) {  
 f = (d & b) | ((~d) & c);  
 g = (5\*i + 1) % 16;  
 } **else if** (i < 48) {  
 f = b ^ c ^ d;  
 g = (3\*i + 5) % 16;  
 } **else** {  
 f = c ^ (b | (~d));  
 g = (7\*i) % 16;  
 }  
 uint32\_t temp = d;  
 d = c;  
 c = b;  
 b = b + LEFTROTATE((a + f + md5\_k[i] + block[g]), md5\_round\_shifts[i]);  
 a = temp;  
 }  
  
 \*h0 += a;  
 \*h1 += b;  
 \*h2 += c;  
 \*h3 += d;  
}

Фрагмент файла Main.cpp с вызовом хэш-функции md5

**if**(config.type == *MD5*){  
 printf("%s\n", "MD5");  
 uint32\_t\* buf32 = (uint32\_t \*)malloc(**sizeof**(uint32\_t)\*16); //512bit buffer  
 uint32\_t hash0 = md5\_a0;  
 uint32\_t hash1 = md5\_b0;  
 uint32\_t hash2 = md5\_c0;  
 uint32\_t hash3 = md5\_d0;  
  
 uint64\_t totalReadBytes = 0;  
 uint64\_t chunkReadBytes = 0;  
 uint64\_t chunkNum = 0;  
 **bool** extrastep = **false**;  
 **do**{  
 chunkReadBytes = fread(buf32, **sizeof**(uint32\_t),16,inputFile)\***sizeof**(uint32\_t);  
 totalReadBytes += chunkReadBytes;  
 printf("\nRead chunk: %llu bytes: %llu totalRead: %llu of: %llu\n",chunkNum,chunkReadBytes,totalReadBytes,inputFileSize);  
 **if**(totalReadBytes==inputFileSize){  
 printf("\nFile Read\n");  
 **if**(chunkReadBytes<=56){  
 uint64\_t extrazeroes = (56-chunkReadBytes)/ **sizeof**(uint32\_t);  
 printf("\nLess than 56, needs extra padding needs %d\n",extrazeroes);  
 uint64\_t newLen;  
 **for**(newLen = chunkReadBytes\*8 + 1; newLen%512!=448; newLen++);  
 newLen /= 8;  
 uint8\_t\* newBuf = NULL;  
 newBuf = (uint8\_t\*) calloc(newLen + 64, 1);  
 memcpy(newBuf, buf32, chunkReadBytes);  
 newBuf[chunkReadBytes] = 128;  
 uint32\_t bitsLen = 8 \* inputFileSize;  
 memcpy(newBuf + newLen, &bitsLen,4);  
 memcpy(buf32,newBuf,64);  
 free(newBuf);  
 }**else**{  
 extrastep = **true**;  
 printf("\nMore than 56, needs extra step\n");  
 }  
 }  
 //expand buf  
 md5\_process\_block(buf32,&hash0,&hash1,&hash2,&hash3);  
 chunkNum++;  
 }**while**(totalReadBytes<inputFileSize);  
 //print md5 hash  
 uint8\_t \*p;  
 p=(uint8\_t \*)&hash0;  
 printf("%2.2x%2.2x%2.2x%2.2x", p[0], p[1], p[2], p[3], hash0);  
 p=(uint8\_t \*)&hash1;  
 printf("%2.2x%2.2x%2.2x%2.2x", p[0], p[1], p[2], p[3], hash1);  
 p=(uint8\_t \*)&hash2;  
 printf("%2.2x%2.2x%2.2x%2.2x", p[0], p[1], p[2], p[3], hash2);  
 p=(uint8\_t \*)&hash3;  
 printf("%2.2x%2.2x%2.2x%2.2x", p[0], p[1], p[2], p[3], hash3);  
 puts("");  
 free(buf32);