**Министерство науки и высшего образования РФ**

Федеральное государственное бюджетное общеобразовательное учреждение высшего образования

«Чувашский Государственный Университет им. И.Н.Ульянова»

Факультет информатики и вычислительной техники

Кафедра компьютерных технологий

Отчет Лабораторной работы №2

По дисциплине: «Информационная безопасность»

Вариант 4

Выполнил студент группы КТ-43-21

Казаков А.Ю.

Проверил ст. преподаватель:

Мытникова Е. А.

Чебоксары, 2025

**Лабораторная работа №3**

Задание: Исследование влияния длины входных данных на хэш: Провести эксперимент, как длина входных данных влияет на время вычисления хэш-функции.

#include <iostream>

#include <string>

#include <vector>

#include <sstream>

#include <iomanip>

#include <chrono>

using namespace std;

// Побитовые операции для SHA-1

#define ROTLEFT(a, b) ((a << b) | (a >> (32 - b)))

// Функция для дополнения исходного сообщения

vector<uint8\_t> padMessage(const string& message) {

vector<uint8\_t> paddedMessage(message.begin(), message.end());

// Добавляем 1 бит (0x80)

paddedMessage.push\_back(0x80);

// Вычисляем необходимый размер дополнения

size\_t originalBitLength = message.size() \* 8;

while ((paddedMessage.size() \* 8) % 512 != 448) {

paddedMessage.push\_back(0x00);

}

// Добавляем исходную длину сообщения в битах (64-битное представление)

for (int i = 7; i >= 0; i--) {

paddedMessage.push\_back(static\_cast<uint8\_t>((originalBitLength >> (i \* 8)) & 0xFF));

}

return paddedMessage;

}

// Основная функция SHA-1

string sha1(const string& message) {

// Инициализируем буфер

uint32\_t h0 = 0x67452301;

uint32\_t h1 = 0xEFCDAB89;

uint32\_t h2 = 0x98BADCFE;

uint32\_t h3 = 0x10325476;

uint32\_t h4 = 0xC3D2E1F0;

// Подготовка данных

vector<uint8\_t> paddedMessage = padMessage(message);

// Разбиваем на 512-битные блоки

for (size\_t chunk = 0; chunk < paddedMessage.size(); chunk += 64) {

uint32\_t w[80] = { 0 };

// Загружаем 16 слов по 32 бита

for (int i = 0; i < 16; i++) {

w[i] = (paddedMessage[chunk + i \* 4] << 24) |

(paddedMessage[chunk + i \* 4 + 1] << 16) |

(paddedMessage[chunk + i \* 4 + 2] << 8) |

(paddedMessage[chunk + i \* 4 + 3]);

}

// Генерация оставшихся 64 слов

for (int i = 16; i < 80; i++) {

w[i] = ROTLEFT((w[i - 3] ^ w[i - 8] ^ w[i - 14] ^ w[i - 16]), 1);

}

// Инициализация переменных для этого блока

uint32\_t a = h0;

uint32\_t b = h1;

uint32\_t c = h2;

uint32\_t d = h3;

uint32\_t e = h4;

// Основной цикл SHA-1

for (int i = 0; i < 80; i++) {

uint32\_t f, k;

if (i < 20) {

f = (b & c) | ((~b) & d);

k = 0x5A827999;

}

else if (i < 40) {

f = b ^ c ^ d;

k = 0x6ED9EBA1;

}

else if (i < 60) {

f = (b & c) | (b & d) | (c & d);

k = 0x8F1BBCDC;

}

else {

f = b ^ c ^ d;

k = 0xCA62C1D6;

}

uint32\_t temp = ROTLEFT(a, 5) + f + e + k + w[i];

e = d;

d = c;

c = ROTLEFT(b, 30);

b = a;

a = temp;

}

// Обновляем итоговые значения

h0 += a;

h1 += b;

h2 += c;

h3 += d;

h4 += e;

}

// Формируем строку хэша

stringstream ss;

ss << hex << setfill('0') << setw(8) << h0

<< setw(8) << h1

<< setw(8) << h2

<< setw(8) << h3

<< setw(8) << h4;

return ss.str();

}

int main() {

setlocale(LC\_ALL, "");

auto start = std::chrono::high\_resolution\_clock::now();

string hash = sha1("Hello, world!");

auto stop = std::chrono::high\_resolution\_clock::now();

auto duration = std::chrono::duration\_cast<std::chrono::microseconds>(stop - start);

auto start\_second = std::chrono::high\_resolution\_clock::now();

string hash\_second = sha1("Lorem Ipsum is simply dummy text of the printing and typesetting industry.");

auto stop\_second = std::chrono::high\_resolution\_clock::now();

auto duration\_second = std::chrono::duration\_cast<std::chrono::microseconds>(stop\_second - start\_second);

cout << "#1 SHA-1: " << hash << endl;

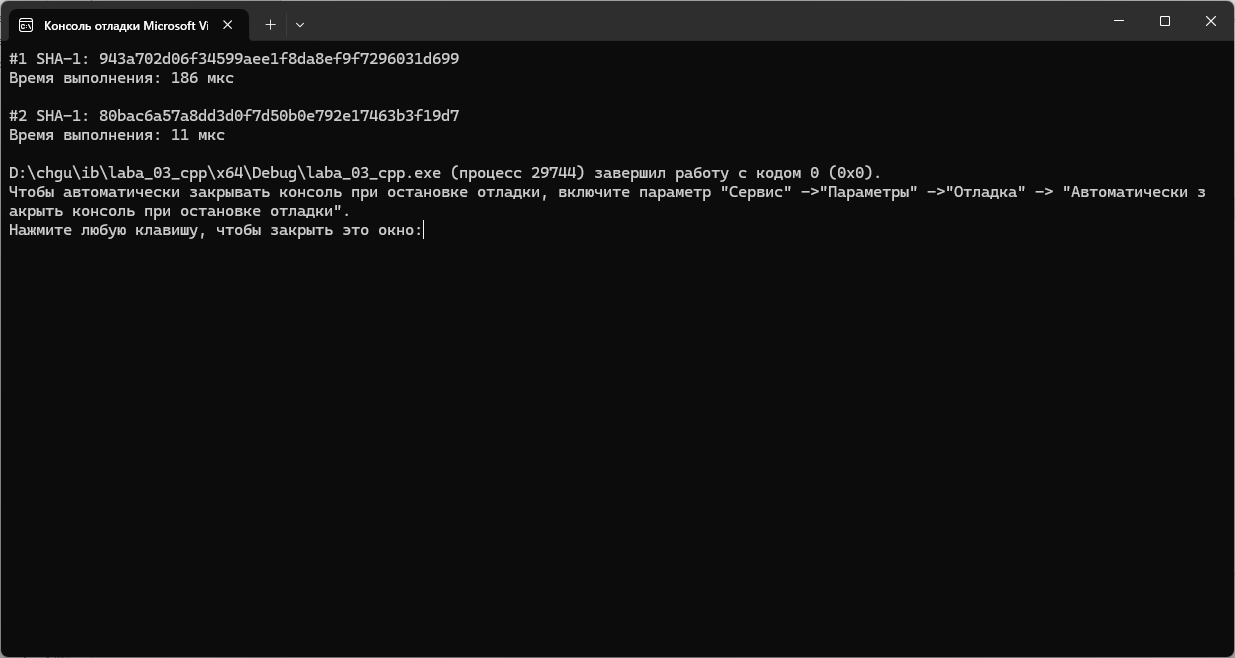
cout << "Время выполнения: " << duration.count() << " мкс\n" << endl;

cout << "#2 SHA-1: " << hash\_second << endl;

cout << "Время выполнения: " << duration\_second.count() << " мкс" << endl;

return 0;

}



**Вывод:** Длина входных данных влияет на время выполнения хеш-функции. Если в хеш-функцию передать длинный текст (>= 64 байт), то хеш-функции не потребуется увеличивать длину текста до 64 байт и функция может сразу приступить к хешированию, если текст < 64 байт, тогда требуется увеличить текст до длины = 64 байт и тем самым увеличивается время выполнения алгоритма хеширования.