МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ государственное БЮДЖЕТНОЕ

образовательное учреждение

высшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кафедра защиты информации



**ПРАКТИЧЕСКАЯ РАБОТА №2**

**«**Изучение основ криптографии**»**

**по дисциплине: «*Программирование*»**

|  |  |
| --- | --- |
| Выполнил:  Студент гр. «АБc-322», «АВТФ»  *Аврамчук Илья Вячеславович*  «7» мая 2024г  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) | Проверил:  Доцент кафедры ЗИ  *Архипова Анастасия Борисовна*  «\_\_\_» \_\_\_\_\_\_ 2024г  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) |

Новосибирск 2024

**Цели и задачи работы:** изучение циклических алгоритмов, операторов цикла, программирование циклического вычислительного процесса на примере математических методов основ криптографии.

**Задание к работе:** Реализовать циклический вычислительный процесс. Самостоятельно решить задачи в соответствии с индивидуальным вариантом.

**Методика выполнения работы:**

1. Разработать алгоритм решения задачи по индивидуальному заданию.

2. Написать и отладить программу решения задачи.

3. Протестировать работу программы на различных исходных данных.

4. Ответить на вопросы по выполненным заданиям, по запросу преподавателя модифицировать код.

5. Ответить на вопросы согласно списку понятий к защите практики (с численными примерами).

6. По запросу преподавателя решить практическое задание на тему «Изучение математических основ криптографии».

**Задание 1.** Реализовать a x mod p Сравнения по модулю простого числа через теорему Ферма и свойства сравнений. Программа должна проверять условия выполнения теоремы Ферма и простоту вводимого пользователем модуля. Реализовать алгоритм через разложение степени в двоичный вид (логарифм).

**Задание 2.** Реализовать обобщенный алгоритм Евклида для вычисления с\*d mod m=1.

**Задание 3.** Реализовать расширенный алгоритм Евклида для вычисления взаимообратного числа с -1 mod m=d.

**Задание 4.** Написать программу, использующую алгоритм шифрования данных для преобразования исходного текста3.

**Вариант 1** - Диффи-Хеллмана

**Вариант 2** - Шамира

**Вариант 3** - Эль-Гамаля

**Вариант 4** – RSA

**Вариант 5** - Хьюза (Hughes)

**Задание 5**. Найти последнюю цифру «трехэтажного числа». Например, 37^8 .

**Задание 6.** Написать сообщение на тему «Атака посередине». Можно продемонстрировать эмуляцию атаки на базе программы задания 4 данной практической работы.

**Задание 7.** Написать сообщение на тему «Стандарты современной криптографии в РФ».

**Задание 1:**

**С++:**

#include <iostream>

#include <cmath> // Для использования математических функций

using namespace std;

// Функция для проверки, является ли число простым

bool isPrime(int number) {

if (number <= 1) return false; // Если число меньше или равно 1, то оно не является простым

for (int i = 2; i <= sqrt(number); ++i) { // Проверяем делители числа до квадратного корня из числа

if (number % i == 0) return false; // Если найден делитель, то число не является простым

}

return true; // Если не найдено делителей, то число простое

}

// Функция для быстрого возведения в степень по модулю

int modularExponentiation(int base, int exponent, int modulus) {

int result = 1; // Инициализируем результат

base = base % modulus; // Уменьшаем base до модуля

while (exponent > 0) { // Пока степень больше 0

if (exponent % 2 == 1) { // Если степень нечетная

result = (result \* base) % modulus; // Умножаем результат на base и берем по модулю

}

exponent = exponent >> 1; // Уменьшаем степень вдвое

base = (base \* base) % modulus; // Умножаем base на само себя и берем по модулю

}

return result; // Возвращаем результат

}

// Функция для проверки выполнения условия теоремы Ферма

bool fermatCondition(int base, int modulus) {

if (modulus <= 1) return false; // Если modulus меньше или равен 1, то условие теоремы Ферма не выполнено

return modularExponentiation(base, modulus - 1, modulus) == 1; // Вычисляем a^(p - 1) % p и сравниваем с 1

}

// Функция для вычисления a^x mod p по первому методу

int firstMethod(int base, int exponent, int modulus) {

if (base % modulus == 0 || !isPrime(modulus) || exponent <= 0) { // Проверяем условия для использования первого метода

cout << "Ошибка!" << endl; // Выводим сообщение об ошибке

return -1; // Возвращаем -1

}

int result = 1; // Инициализируем результат

for (int i = 1; i <= exponent; ++i) { // Цикл для возведения числа в степень

result \*= base; // Умножаем результат на число

result %= modulus; // Берем по модулю

}

return result; // Возвращаем результат

}

// Функция для вычисления a^x mod p по второму методу

int secondMethod(int base, int exponent, int modulus) {

if (!isPrime(modulus)) { // Проверяем модуль на простоту

cout << "Модуль должен быть простым числом!" << endl; // Выводим сообщение об ошибке

return -1; // Возвращаем -1

}

if (base % modulus == 0 && exponent > 1) { // Проверяем условие теоремы Ферма

cout << "Условие теоремы не выполнено" << endl; // Выводим сообщение об ошибке

return -1; // Возвращаем -1

}

int t = (int) (log(exponent) / log(2)); // Вычисляем количество степеней

int binaryExponent[t + 1]; // Создаем массив для хранения двоичного представления степени

int origExponent = exponent; // Сохраняем оригинальную степень

for (int i = t; i >= 0; --i) { // Цикл для заполнения массива двоичным представлением степени

binaryExponent[i] = origExponent % 2; // Заполняем массив остатками от деления степени на 2

origExponent /= 2; // Делим степень на 2

}

int remainders[t + 1]; // Создаем массив для хранения остатков от деления

remainders[0] = base; // Записываем начальное значение

int temp = base; // Сохраняем начальное значение

for (int i = 1; i <= t; ++i) { // Цикл для вычисления остатков

temp = (temp \* temp) % modulus; // Вычисляем квадрат и берем по модулю

remainders[i] = temp; // Записываем остаток в массив

}

int result = 1; // Инициализируем результат

for (int i = t; i >= 0; --i) { // Обратный проход по массиву двоичного представления степени

if (binaryExponent[i] == 1) { // Если соответствующий бит равен 1

result = (result \* remainders[t - i]) % modulus; // Умножаем результат на соответствующий остаток и берем по модулю

}

}

return result; // Возвращаем результат

}

int main() { // Главная функция программы

int base, exponent, modulus; // Переменные для хранения введенных чисел

cout << "Введите число a: "; // Выводим приглашение для ввода числа

cin >> base; // Считываем число

cout << "Введите степень x: "; // Выводим приглашение для ввода степени

cin >> exponent; // Считываем степень

cout << "Введите простое число p: "; // Выводим приглашение для ввода простого числа

cin >> modulus; // Считываем простое число

int resultFirstMethod = firstMethod(base, exponent, modulus); // Вычисляем результат первого метода

cout << "Первый метод" << endl; // Выводим сообщение о первом методе

cout << "Остаток: " << resultFirstMethod << endl; // Выводим результат первого метода

int resultSecondMethod = secondMethod(base, exponent, modulus); // Вычисляем результат второго метода

if (resultSecondMethod != -1) { // Если результат не равен -1

cout << "Второй метод" << endl; // Выводим сообщение о втором методе

cout << "Остаток: " << resultSecondMethod << endl; // Выводим результат второго метода

}

return 0; // Возвращаем 0, указывая на успешное завершение программы

}

**Java:**

import java.util.Scanner;

public class Main {

// Функция для проверки, является ли число простым

public static boolean isPrime(int number) {

if (number <= 1) return false; // Если число меньше или равно 1, то оно не является простым

for (int i = 2; i <= Math.sqrt(number); ++i) { // Проверяем делители числа до квадратного корня из числа

if (number % i == 0) return false; // Если найден делитель, то число не является простым

}

return true; // Если не найдено делителей, то число простое

}

// Функция для быстрого возведения в степень по модулю

public static int modularExponentiation(int base, int exponent, int modulus) {

int result = 1; // Инициализируем результат

base = base % modulus; // Уменьшаем base до модуля

while (exponent > 0) { // Пока степень больше 0

if (exponent % 2 == 1) { // Если степень нечетная

result = (result \* base) % modulus; // Умножаем результат на base и берем по модулю

}

exponent = exponent >> 1; // Уменьшаем степень вдвое

base = (base \* base) % modulus; // Умножаем base на само себя и берем по модулю

}

return result; // Возвращаем результат

}

// Функция для проверки выполнения условия теоремы Ферма

public static boolean fermatCondition(int base, int modulus) {

if (modulus <= 1) return false; // Если modulus меньше или равен 1, то условие теоремы Ферма не выполнено

return modularExponentiation(base, modulus - 1, modulus) == 1; // Вычисляем a^(p - 1) % p и сравниваем с 1

}

// Функция для вычисления a^x mod p по первому методу

public static int firstMethod(int base, int exponent, int modulus) {

if (base % modulus == 0 || !isPrime(modulus) || exponent <= 0) { // Проверяем условия для использования первого метода

System.out.println("Ошибка!"); // Выводим сообщение об ошибке

return -1; // Возвращаем -1

}

int result = 1; // Инициализируем результат

for (int i = 1; i <= exponent; ++i) { // Цикл для возведения числа в степень

result \*= base; // Умножаем результат на число

result %= modulus; // Берем по модулю

}

return result; // Возвращаем результат

}

// Функция для вычисления a^x mod p по второму методу

public static int secondMethod(int base, int exponent, int modulus) {

if (!isPrime(modulus)) { // Проверяем модуль на простоту

System.out.println("Модуль должен быть простым числом!"); // Выводим сообщение об ошибке

return -1; // Возвращаем -1

}

if (base % modulus == 0 && exponent > 1) { // Проверяем условие теоремы Ферма

System.out.println("Условие теоремы не выполнено"); // Выводим сообщение об ошибке

return -1; // Возвращаем -1

}

int t = (int) (Math.log(exponent) / Math.log(2)); // Вычисляем количество степеней

int[] binaryExponent = new int[t + 1]; // Создаем массив для хранения двоичного представления степени

int origExponent = exponent; // Сохраняем оригинальную степень

for (int i = t; i >= 0; --i) { // Цикл для заполнения массива двоичным представлением степени

binaryExponent[i] = origExponent % 2; // Заполняем массив остатками от деления степени на 2

origExponent /= 2; // Делим степень на 2

}

int[] remainders = new int[t + 1]; // Создаем массив для хранения остатков от деления

remainders[0] = base; // Записываем начальное значение

int temp = base; // Сохраняем начальное значение

for (int i = 1; i <= t; ++i) { // Цикл для вычисления остатков

temp = (temp \* temp) % modulus; // Вычисляем квадрат и берем по модулю

remainders[i] = temp; // Записываем остаток в массив

}

int result = 1; // Инициализируем результат

for (int i = binaryExponent.length - 1; i >= 0; --i) { // Обратный проход по массиву двоичного представления степени

if (binaryExponent[i] == 1) { // Если соответствующий бит равен 1

result = (result \* remainders[binaryExponent.length - 1 - i]) % modulus; // Умножаем результат на соответствующий остаток и берем по модулю

}

}

return result; // Возвращаем результат

}

public static void main(String[] args) { // Главная функция программы

Scanner scanner = new Scanner(System.in); // Создаем объект Scanner для ввода данных

System.out.print("Введите число a: "); // Выводим приглашение для ввода числа

int base = scanner.nextInt(); // Считываем число

System.out.print("Введите степень x: "); // Выводим приглашение для ввода степени

int exponent = scanner.nextInt(); // Считываем степень

System.out.print("Введите простое число p: "); // Выводим приглашение для ввода простого числа

int modulus = scanner.nextInt(); // Считываем простое число

int resultFirstMethod = firstMethod(base, exponent, modulus); // Вычисляем результат первого метода

System.out.println("Первый метод"); // Выводим сообщение о первом методе

System.out.println("Остаток: " + resultFirstMethod); // Выводим результат первого метода

int resultSecondMethod = secondMethod(base, exponent, modulus); // Вычисляем результат второго метода

if (resultSecondMethod != -1) { // Если результат не равен -1

System.out.println("Второй метод"); // Выводим сообщение о втором методе

System.out.println("Остаток: " + resultSecondMethod); // Выводим результат второго метода

}

}

}

**Задание 2:**

**С++:**

#include <iostream>

using namespace std;

// Структура для представления пары целых чисел

struct Pair {

int first, second; // Поля для первого и второго элементов пары

// Конструктор структуры Pair

Pair(int first, int second) : first(first), second(second) {}

};

// Функция для нахождения модулярного обратного числа

Pair modInverse(int c, int m) {

Pair u(m, 0); // Инициализируем пару (m, 0)

Pair v(c, 1); // Инициализируем пару (c, 1)

while (v.first) { // Пока второй элемент пары v не равен нулю

int q = u.first / v.first; // Находим частное от деления первого элемента пары u на первый элемент пары v

Pair t(u.first % v.first, u.second - q \* v.second); // Вычисляем новую пару (u % v, u - q \* v)

u = v; // Обновляем пару u

v = t; // Обновляем пару v

}

if (u.second < 0) // Если второй элемент пары u отрицательный

u.second += m; // Добавляем m к второму элементу пары u

return u; // Возвращаем пару u

}

int main() {

setlocale(LC\_ALL, "russian"); // Устанавливаем локаль для корректного вывода сообщений на русском языке

int c, m;

cout << "Введите c: "; // Выводим приглашение для ввода пользователем c

cin >> c; // Читаем значение c

cout << "Введите m: "; // Выводим приглашение для ввода пользователем m

cin >> m; // Читаем значение m

Pair result = modInverse(c, m); // Вызываем функцию modInverse для нахождения модулярного обратного числа

if (result.first == 1) // Если первый элемент результата равен 1

cout << "d = " << result.second << endl; // Выводим модулярное обратное число d

else

cout << "d не существует, так как c и m - не взаимно простые числа" << endl; // Выводим сообщение о том, что модулярное обратное число не существует

return 0;

}

**Java:**

import java.util.Scanner;

// Класс для представления пары целых чисел

class Pair {

int first, second; // Поля для первого и второго элементов пары

// Конструктор класса Pair

Pair(int first, int second) {

this.first = first;

this.second = second;

}

}

public class Main {

// Функция для нахождения модулярного обратного числа

static Pair modInverse(int c, int m) {

Pair u = new Pair(m, 0); // Инициализируем пару (m, 0)

Pair v = new Pair(c, 1); // Инициализируем пару (c, 1)

while (v.first != 0) { // Пока второй элемент пары v не равен нулю

int q = u.first / v.first; // Находим частное от деления первого элемента пары u на первый элемент пары v

Pair t = new Pair(u.first % v.first, u.second - q \* v.second); // Вычисляем новую пару (u % v, u - q \* v)

u = v; // Обновляем пару u

v = t; // Обновляем пару v

}

if (u.second < 0) // Если второй элемент пары u отрицательный

u.second += m; // Добавляем m к второму элементу пары u

return u; // Возвращаем пару u

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Введите c: "); // Выводим приглашение для ввода пользователем c

int c = scanner.nextInt(); // Читаем значение c

System.out.print("Введите m: "); // Выводим приглашение для ввода пользователем m

int m = scanner.nextInt(); // Читаем значение m

Pair result = modInverse(c, m); // Вызываем функцию modInverse для нахождения модулярного обратного числа

if (result.first == 1) // Если первый элемент результата равен 1

System.out.println("d = " + result.second); // Выводим модулярное обратное число d

else

System.out.println("d не существует, так как c и m - не взаимно простые числа"); // Выводим сообщение о том, что модулярное обратное число не существует

}

}

**Задание 3:**

**С++:**

#include <iostream>

using namespace std;

// Структура для представления тройки целых чисел (d, a, b)

struct Triple {

int first, second, third;

Triple(int d, int a, int b) : first(d), second(a), third(b) {}

};

// Функция расширенного алгоритма Евклида

// Возвращает тройку (d, a, b), где d - НОД(a, b), а также находит коэффициенты x и y такие, что a\*x + b\*y = d

Triple extendedEuclidean(int a, int b) {

if (b == 0) {

return Triple(a, 1, 0); // Базовый случай: если b равно 0, возвращаем (a, 1, 0)

} else {

Triple vals = extendedEuclidean(b, a % b); // Рекурсивный случай: вызываем функцию для b и остатка от деления a на b

int d = vals.first; // НОД(a, b)

int x = vals.third; // Коэффициент перед a в уравнении a\*x + b\*y = d

int y = vals.second - (a / b) \* vals.third; // Коэффициент перед b в уравнении a\*x + b\*y = d

return Triple(d, x, y);

}

}

// Функция для вычисления взаимнообратного числа s^-1 mod m

int modInverse(int s, int m) {

Triple vals = extendedEuclidean(s, m); // Используем расширенный алгоритм Евклида

int d = vals.first; // НОД(s, m)

int a = vals.second; // Коэффициент перед s в уравнении s\*a + m\*b = d

if (d > 1) {

return -1; // Если НОД(s, m) больше 1, взаимнообратное число не существует

}

return (a % m + m) % m; // Возвращаем коэффициент a, удовлетворяющий условию a\*s + b\*m = 1

}

int main() {

setlocale(LC\_ALL, "russian");

int s, m;

cout << "Введите c: ";

cin >> s; // Число, для которого нужно найти взаимнообратное

cout << "Введите m: ";

cin >> m; // Модуль

int inverse = modInverse(s, m); // Вызываем функцию modInverse

if (inverse == -1) {

cout << "Обратного числа не существует, так как числа не взаимно простые." << endl;

} else {

cout << "Обратное число для " << s << " по модулю " << m << " = " << inverse << endl;

}

return 0;

}

**Java:**

import java.util.Scanner;

// Класс для представления тройки целых чисел (d, a, b)

class Triple {

int first, second, third;

// Конструктор класса Triple

Triple(int d, int a, int b) {

this.first = d;

this.second = a;

this.third = b;

}

}

public class Main {

// Функция расширенного алгоритма Евклида

// Возвращает тройку (d, a, b), где d - НОД(a, b),

// а также находит коэффициенты x и y такие, что a\*x + b\*y = d

static Triple extendedEuclidean(int a, int b) {

if (b == 0) {

return new Triple(a, 1, 0); // Базовый случай: если b равно 0, возвращаем (a, 1, 0)

} else {

// Рекурсивный случай: вызываем функцию для b и остатка от деления a на b,

// получаем результаты (НОД, x, y) и вычисляем новые коэффициенты

Triple vals = extendedEuclidean(b, a % b);

int d = vals.first; // НОД(a, b)

int x = vals.third; // Коэффициент перед a в уравнении a\*x + b\*y = d

int y = vals.second - (a / b) \* vals.third; // Коэффициент перед b в уравнении a\*x + b\*y = d

return new Triple(d, x, y);

}

}

// Функция для вычисления взаимнообратного числа s^-1 mod m

static int modInverse(int s, int m) {

// Используем расширенный алгоритм Евклида для нахождения НОД(s, m) и коэффициентов a и b

Triple vals = extendedEuclidean(s, m);

int d = vals.first; // НОД(s, m)

int a = vals.second; // Коэффициент перед s в уравнении s\*a + m\*b = d

if (d > 1) {

return -1; // Если НОД(s, m) больше 1, взаимнообратное число не существует

}

return (a % m + m) % m; // Возвращаем коэффициент a, удовлетворяющий условию a\*s + b\*m = 1

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Введите c: ");

int s = scanner.nextInt(); // Число, для которого нужно найти взаимнообратное

System.out.print("Введите m: ");

int m = scanner.nextInt(); // Модуль

int inverse = modInverse(s, m); // Вызываем функцию modInverse

if (inverse == -1) {

System.out.println("Обратного числа не существует, так как числа не взаимно простые.");

} else {

System.out.println("Обратное число для " + s + " по модулю " + m + " = " + inverse);

}

}

}

**Задание 4:(Диффи-Хеллман)**

**С++:**

#include <iostream>

using namespace std;

// Функция для вычисления возведения числа в степень по модулю

int modPow(int base, int exp, int mod) {

int result = 1; // Инициализация результата

base %= mod; // Получение остатка от деления базы на модуль

while (exp > 0) { // Цикл, выполняющийся, пока показатель степени больше нуля

if (exp % 2 == 1) { // Проверка на нечетность показателя степени

result = (result \* base) % mod; // Обновление результата

}

base = (base \* base) % mod; // Обновление базы

exp /= 2; // Уменьшение показателя степени в два раза

}

return result; // Возврат результата

}

int main() {

// Шаг 1: Генерация общих параметров

int p, g; // Объявление переменных для простого числа p и примитивного корня g

cout << "Введите простое число p: "; // Вывод на экран запроса ввода простого числа p

cin >> p; // Ввод значения простого числа p

cout << "Введите примитивный корень по модулю p (g): "; // Вывод на экран запроса ввода примитивного корня g

cin >> g; // Ввод значения примитивного корня g

// Шаг 2: Генерация секретных ключей

int a, b; // Объявление переменных для секретных ключей Alice и Bob

cout << "Alice: Введите свой секретный ключ a: "; // Вывод на экран запроса ввода секретного ключа Alice

cin >> a; // Ввод значения секретного ключа Alice

cout << "Bob: Введите свой секретный ключ b: "; // Вывод на экран запроса ввода секретного ключа Bob

cin >> b; // Ввод значения секретного ключа Bob

// Шаг 3: Вычисление открытых ключей

int A = modPow(g, a, p); // Вычисление открытого ключа Alice

int B = modPow(g, b, p); // Вычисление открытого ключа Bob

// Шаг 4: Обмен и вычисление общего секретного ключа

int secretKeyA = modPow(B, a, p); // Вычисление общего секретного ключа Alice

int secretKeyB = modPow(A, b, p); // Вычисление общего секретного ключа Bob

// Проверка совпадения общих секретных ключей

if (secretKeyA == secretKeyB) { // Проверка совпадения общих секретных ключей Alice и Bob

cout << "Alice и Bob согласовали общий секретный ключ: " << secretKeyA << endl; // Вывод сообщения о согласовании общего секретного ключа

} else {

cout << "Ошибка: общие секретные ключи не совпадают." << endl; // Вывод сообщения об ошибке

}

return 0; // Возврат нуля, сигнализирующего об успешном завершении программы

}

**Java:**

import java.util.Scanner;

public class Main {

// Функция для вычисления возведения числа в степень по модулю

static int modPow(int base, int exp, int mod) {

int result = 1; // Инициализация результата

base %= mod; // Получение остатка от деления базы на модуль

while (exp > 0) { // Цикл, выполняющийся, пока показатель степени больше нуля

if (exp % 2 == 1) { // Проверка на нечетность показателя степени

result = (result \* base) % mod; // Обновление результата

}

base = (base \* base) % mod; // Обновление базы

exp /= 2; // Уменьшение показателя степени в два раза

}

return result; // Возврат результата

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Шаг 1: Генерация общих параметров

System.out.print("Введите простое число p: "); // Вывод на экран запроса ввода простого числа p

int p = scanner.nextInt(); // Ввод значения простого числа p

System.out.print("Введите примитивный корень по модулю p (g): "); // Вывод на экран запроса ввода примитивного корня g

int g = scanner.nextInt(); // Ввод значения примитивного корня g

// Шаг 2: Генерация секретных ключей

System.out.print("Alice: Введите свой секретный ключ a: "); // Вывод на экран запроса ввода секретного ключа Alice

int a = scanner.nextInt(); // Ввод значения секретного ключа Alice

System.out.print("Bob: Введите свой секретный ключ b: "); // Вывод на экран запроса ввода секретного ключа Bob

int b = scanner.nextInt(); // Ввод значения секретного ключа Bob

// Шаг 3: Вычисление открытых ключей

int A = modPow(g, a, p); // Вычисление открытого ключа Alice

int B = modPow(g, b, p); // Вычисление открытого ключа Bob

// Шаг 4: Обмен и вычисление общего секретного ключа

int secretKeyA = modPow(B, a, p); // Вычисление общего секретного ключа Alice

int secretKeyB = modPow(A, b, p); // Вычисление общего секретного ключа Bob

// Проверка совпадения общих секретных ключей

if (secretKeyA == secretKeyB) { // Проверка совпадения общих секретных ключей Alice и Bob

System.out.println("Alice и Bob согласовали общий секретный ключ: " + secretKeyA); // Вывод сообщения о согласовании общего секретного ключа

} else {

System.out.println("Ошибка: общие секретные ключи не совпадают."); // Вывод сообщения об ошибке

}

scanner.close(); // Закрытие сканнера

}

}

**Задание 5:**

**С++:**

#include <iostream>

using namespace std;

class LastDigitOfPower {

private:

int Osnovanie; // переменная для хранения основания степени

int Stepen; // переменная для хранения показателя степени

public:

LastDigitOfPower(int Osnovanie, int Stepen) : Osnovanie(Osnovanie), Stepen(Stepen) {} // конструктор класса с инициализацией полей

int lastDigitOfPower() { // метод для определения последней цифры степени

if (Osnovanie == 0) { // проверка, если основание равно нулю

return 0; // возвращаем ноль

}

int lastDigit = Osnovanie % 10; // определяем последнюю цифру основания

int period = getPeriodOfLastDigit(lastDigit); // получаем период последней цифры

int remainder = Stepen % period; // определяем остаток от деления показателя степени на период

if (remainder == 0) { // если остаток равен нулю

remainder = period; // присваиваем остатку значение периода

}

int result = 1; // переменная для хранения результата

for (int i = 0; i < remainder; i++) { // цикл от 0 до остатка

result = (result \* lastDigit) % 10; // вычисляем результат, учитывая только последнюю цифру

}

return result; // возвращаем результат

}

private:

static int getPeriodOfLastDigit(int lastDigit) { // статический метод для определения периода последней цифры

if (lastDigit == 1 || lastDigit == 5) { // если последняя цифра равна 1 или 5

return 1; // период равен 1

}

else { // иначе

return 4; // период равен 4

}

}

};

int main() { // главная функция программы

setlocale(LC\_ALL, "russian"); // установка локали для вывода на русском языке

int Osnovanie, Stepen1, Stepen2; // объявление переменных для основания и показателей степени

cout << "Введите основание: "; // вывод приглашения на ввод основания

cin >> Osnovanie; // считываем основание с клавиатуры

cout << "Введите показатель 1: "; // вывод приглашения на ввод первого показателя степени

cin >> Stepen1; // считываем первый показатель степени с клавиатуры

cout << "Введите показатель 2: "; // вывод приглашения на ввод второго показателя степени

cin >> Stepen2; // считываем второй показатель степени с клавиатуры

LastDigitOfPower obj(Stepen1, Stepen2); // создаем объект класса с передачей показателей степени

int lastDigit = obj.lastDigitOfPower(); // вызываем метод для определения последней цифры степени

LastDigitOfPower obj2(Osnovanie, lastDigit); // создаем второй объект класса с передачей основания и последней цифры предыдущей степени

int lastDigit2 = obj2.lastDigitOfPower(); // вызываем метод для определения последней цифры степени

cout << "Последняя цифра данного числа: " << lastDigit2 << endl; // выводим результат

return 0; // возвращаем ноль, указывая на успешное завершение программы

}

**Java:**

import java.util.Scanner;

class LastDigitOfPower {

private int Osnovanie; // переменная для хранения основания степени

private int Stepen; // переменная для хранения показателя степени

public LastDigitOfPower(int Osnovanie, int Stepen) {

this.Osnovanie = Osnovanie;

this.Stepen = Stepen;

}

public int lastDigitOfPower() {

if (Osnovanie == 0) { // проверка, если основание равно нулю

return 0; // возвращаем ноль

}

int lastDigit = Osnovanie % 10; // определяем последнюю цифру основания

int period = getPeriodOfLastDigit(lastDigit); // получаем период последней цифры

int remainder = Stepen % period; // определяем остаток от деления показателя степени на период

if (remainder == 0) { // если остаток равен нулю

remainder = period; // присваиваем остатку значение периода

}

int result = 1; // переменная для хранения результата

for (int i = 0; i < remainder; i++) { // цикл от 0 до остатка

result = (result \* lastDigit) % 10; // вычисляем результат, учитывая только последнюю цифру

}

return result; // возвращаем результат

}

private static int getPeriodOfLastDigit(int lastDigit) { // статический метод для определения периода последней цифры

if (lastDigit == 1 || lastDigit == 5) { // если последняя цифра равна 1 или 5

return 1; // период равен 1

} else { // иначе

return 4; // период равен 4

}

}

}

public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Введите основание: ");

int Osnovanie = scanner.nextInt();

System.out.print("Введите показатель 1: ");

int Stepen1 = scanner.nextInt();

System.out.print("Введите показатель 2: ");

int Stepen2 = scanner.nextInt();

LastDigitOfPower obj = new LastDigitOfPower(Stepen1, Stepen2);

int lastDigit = obj.lastDigitOfPower();

LastDigitOfPower obj2 = new LastDigitOfPower(Osnovanie, lastDigit);

int lastDigit2 = obj2.lastDigitOfPower();

System.out.println("Последняя цифра данного числа: " + lastDigit2);

}

}

**Выводы:**

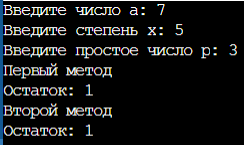
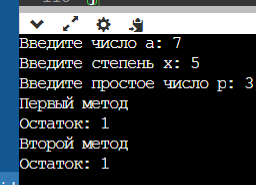
 

Рис.1 Вывод 1 задания (Слева C++, справа Java)

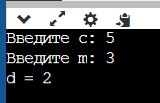
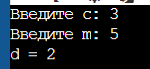


Рис. 2 Вывод 2 задания (Слева C++, справа Java)

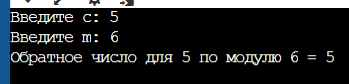
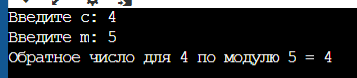


Рис. 3 Вывод 3 задания (Слева C++, справа Java)

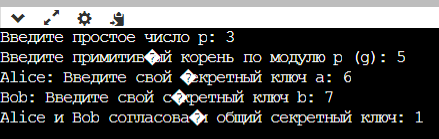
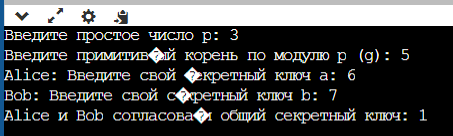


Рис. 4 Вывод 4 задания (Слева C++, справа Java)

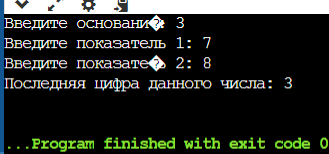
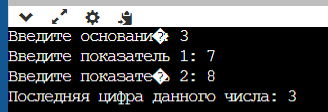


Рис. 5 Вывод 5 задания (Слева C++, справа Java)

**Задание 6:**

****Атака посередине: Опасность, которая грозит вашей безопасности в сети****

Атака посередине - это хакерский метод, который становится все более распространенным и угрожающим в нашем цифровом мире. Эта форма кибератаки основана на том, что злоумышленник встраивается между двумя или более коммуницирующими сторонами и манипулирует передаваемой информацией.

Как это происходит? Когда две стороны, например, пользователь и веб-сервер, обмениваются данными, злоумышленник встраивается в этот обмен. Затем он перехватывает передаваемые данные или даже изменяет их без ведома обеих сторон. Это может привести к серьезным последствиям, таким как украденная конфиденциальная информация, потеря данных, финансовые потери или даже повреждение репутации.

Наиболее беспокоящим аспектом атаки посередине является то, что она часто происходит незаметно. Злоумышленники могут использовать различные методы для встраивания в сетевой обмен, включая подделку IP-адресов, атаку "человек посередине" (Man-in-the-Middle) или вредоносное программное обеспечение.

Чтобы защититься от атаки посередине, необходимо принять соответствующие меры безопасности. Это включает в себя использование шифрования данных, использование надежных сетевых соединений, регулярное обновление программного обеспечения для защиты от известных уязвимостей, а также обучение сотрудников по вопросам кибербезопасности.

Также важно поддерживать бдительность и быть готовыми к распознаванию признаков атаки посередине. Это могут быть необычные запросы на подтверждение личной информации, неожиданные изменения в передаваемых данных или подозрительные запросы на изменение настроек безопасности.

В целом, осведомленность о рисках атаки посередине и регулярное обновление мер безопасности помогут минимизировать угрозу для вашей сетевой безопасности.

**Задание 7:**

Стандарты современной криптографии в Российской Федерации представляют собой фундаментальный элемент в обеспечении безопасности цифровой информации. В современном мире, где данные играют ключевую роль в различных аспектах жизни – от финансовых транзакций до обмена государственной информацией, надежная защита данных становится критически важной.

Эти стандарты криптографии охватывают широкий спектр методов, алгоритмов и протоколов, разработанных для обеспечения конфиденциальности, целостности и аутентичности информации.

Блок 1: Шифрование

ГОСТ 28147-89 (Магма): Этот стандарт определяет алгоритм симметричного блочного шифрования. Шифр "Магма" широко применяется в различных сферах, включая коммерческие и государственные системы связи.

Блок 2: Цифровая подпись

ГОСТ Р 34.10-2012: Стандарт, который определяет алгоритмы цифровой подписи, включая ГОСТ Р 34.10-2012/256, ГОСТ Р 34.10-2012/512 и другие. Эти алгоритмы используются для обеспечения аутентификации и целостности данных.

Блок 3: Хэширование

ГОСТ Р 34.11-2012: Этот стандарт определяет алгоритмы хэширования, такие как ГОСТ Р 34.11-2012/256 и ГОСТ Р 34.11-2012/512. Они применяются для создания криптографических хэш-значений, используемых для проверки целостности данных.

Блок 4: Ключи

ГОСТ Р 34.12-2015 (Кузнечик): Этот стандарт определяет алгоритм блочного шифрования и методы формирования ключей. Кузнечик широко используется в различных системах шифрования и аутентификации.

Криптографические стандарты в России постоянно совершенствуются и адаптируются к изменяющимся условиям и угрозам в области информационной безопасности. Это процесс, включающий в себя как разработку новых алгоритмов, так и анализ существующих на предмет уязвимостей. Такой подход позволяет эффективно защищать информацию как на государственном, так и на корпоративном уровнях.

В контексте стратегии цифрового развития страны, развитие и поддержка стандартов криптографии становятся неотъемлемой частью усилий по обеспечению информационной безопасности. Это необходимо как для защиты чувствительных данных в экономической и финансовой сферах, так и для обеспечения безопасности государственных коммуникаций и обороны. Таким образом, стандарты современной криптографии в России играют ключевую роль в обеспечении цифровой безопасности и сохранности информации в эпоху цифрового прогресса.