Міністерство освіти і науки України
Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення
Дисципліна: Алгоритми та структури даних

Лабораторна робота №3

Тема: «ГЕНЕРАТОРИ ПСЕВДОВИПАДКОВИХ ЧИСЕЛ»

Виконав: ст. гр. КН-24
Куріщенко П. В.
Перевірив: викладач
Константинова Л. В.

Кропивницький 2025

Мета: Навчитися реалізовувати алгоритми генерації псевдовипадкових чисел.

Завдання: Реалізувати розглянуті генератори псевдовипадкових чисел(Парка-Міллера, Л'Екюера, Блюма-Блюма-Шуба).

Лістинг *randomgenerator.h*:

```
#ifndef RANDOMGENERATOR_H
#define RANDOMGENERATOR_H

class RandomGenerator {
protected:
    long long m_seed;

public:
    RandomGenerator(long long seed) : m_seed(seed) {}
    virtual long long next() = 0;
    virtual ~RandomGenerator() {}
};

#endif // RANDOMGENERATOR_H
```

Лістинг *lcg.h*:

```
#ifndef LCG_H
#define LCG_H

#include "randomgenerator.h"

class LCG : public RandomGenerator {
private:
    long long m_multiplier;
    long long m_modulus;
    long long m_increment;

public:
    LCG(long long seed, long long multiplier, long long modulus, long long
increment = 0)
        : RandomGenerator(seed), m_multiplier(multiplier),
m_modulus(modulus), m_increment(increment) {}

    long long next() override {
        m_seed = (m_multiplier * m_seed + m_increment) % m_modulus;
        return m_seed;
    }
};

#endif // LCG_H
```

Лістинг *parkmiller.h*:

```cpp
#ifndef PARKMILLER_H
#define PARKMILLER_H

#include "lcg.h"

class ParkMiller : public RandomGenerator {
private:
    static const long long MULT = 16807;
    static const long long MOD = 2147483647;

    LCG m_lcg;

public:
    ParkMiller(long long seed)
        : RandomGenerator(seed), m_lcg(seed, MULT, MOD) {}

    long long next() override { return m_lcg.next(); }
};

#endif // PARKMILLER_H
```

Лістинг *lecuyer.h*:

```cpp
#ifndef LECUYER_H
#define LECUYER_H

#include "lcg.h"

class LEcuyer : public RandomGenerator {
private:
    static const long long MULT_1 = 40014;
    static const long long MOD_1 = 2147483563;

    static const long long MULT_2 = 40692;
    static const long long MOD_2 = 2147483399;

    LCG m_gen1;
    LCG m_gen2;

public:
    LEcuyer(long long seed1, long long seed2)
        : RandomGenerator(seed1),
        m_gen1(seed1, MULT_1, MOD_1),
        m_gen2(seed2, MULT_2, MOD_2) {}

    long long next() override {
        long long firstNum = m_gen1.next();
        long long secondNum = m_gen2.next();

        long long result = firstNum - secondNum;
        if (result < 0) result += MOD_1;

        return result;
    }
};

#endif // LECUYER_H
```

Лістинг *bbs.h*:

```
#ifndef BBS_H
#define BBS_H

#include <numeric>

#include "randomgenerator.h"

class BBS : public RandomGenerator {
private:
    long long m_modulus;

public:
    BBS(long long seed, long long firstPrime, long long secondPrime)
        : RandomGenerator(seed)
    {
        m_modulus = firstPrime * secondPrime;
        if (seed <= 0 || seed >= m_modulus || std::gcd(seed, m_modulus) != 1)
{
            std::cerr << "Seed має бути в {0, n} і взаємно простим з n.
Використано seed = 3" << std::endl;
            m_seed = 3;
        }

        m_seed = (m_seed * m_seed) % m_modulus;
    }

    long long next() override {
        m_seed = (m_seed * m_seed) % m_modulus;
        return m_seed % 2;
    }

    int nextByte(int bits = 8) {
        int byte = 0;
        for (int count = 0; count < bits; ++count)
            byte = (byte << 1) | next();

        return byte;
    }
};

#endif // BBS_H
```

Лістинг *main.cpp*:

```
#include <iostream>
#include <ctime>

#include "parkmiller.h"
#include "lecuyer.h"
#include "bbs.h"

using namespace std;

void printSequence(RandomGenerator& gen, int numbersAmount, const string&
title) {
    cout << title;
```

```
    for (int count = 0; count < numbersAmount; ++count)
        cout << gen.next() << " ";
    cout << endl;
}

int main() {
    long long seed = time(nullptr);

    ParkMiller pmGen(seed);
    printSequence(pmGen, 5, "Мінімальний генератор Парка-Міллера: ");

    LEcuyer lecGen(seed, seed / 2);
    printSequence(lecGen, 5, "Алгоритм Л'Екюера: ");

    BBS bbsGen(seed % 1000 + 3, 383, 503);

    cout << "Алгоритм Блюма-Блюма-Шуба (біти): ";
    for (int count = 0; count < 32; ++count) {
        cout << bbsGen.next();
        if ((count + 1) % 8 == 0) cout << " ";
    }
    cout << endl;

    cout << "                              (як байти): ";
    for (int count = 0; count < 4; ++count)
        cout << bbsGen.nextByte() << " ";
    cout << endl;

    return 0;
}
```

Результат виконання:

```
Мінімальний генератор Парка-Міллера: 692360470 1436019844 1764293122 20303678 1941499920
Алгоритм Л'Екюера: 492355393 1368583192 520284751 1922426682 1621776587
Алгоритм Блюма-Блюма-Шуба (біти): 10010110 11001111 10001010 01100000
                          (як байти): 66 58 11 177
```

Висновки: У роботі були розглянуті різні типи генераторів: лінійний конгруентний (Park–Miller), комбінований генератор Л’Екюєра та криптографічно стійкий алгоритм Блюма–Блюма–Шуба.

Лінійні генератори є простими та швидкими, але мають обмежені статистичні властивості. Комбіновані методи покращують період і якість послідовності. Алгоритм Блюма–Блюма–Шуба є повільнішим, проте забезпечує вищу криптографічну стійкість.