Учреждение образования

“БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ”

Отчет по лабораторной работе №3 “ Исследование циклического избыточного кода (CRC)”

**Цель работы:** получение навыков использования циклического избыточного кода (CRC)

**Краткие теоретические сведения:** При передаче данных через каналы связи возможны ошибки, искажающие информацию. Для их обнаружения и исправления применяются методы, основанные на избыточности данных. Основные методы обнаружения ошибок:

1. Посимвольный контроль четности (поперечный): Добавляется бит четности к каждому байту. Прост в реализации, но неэффективен при множественных ошибках.
2. Поблочный контроль четности (продольный): Биты четности рассчитываются для блока символов. Лучше выявляет и исправляет ошибки, но пропускает некоторые типы искажений.
3. Контрольные суммы: Сумма значений блока данных. Простой метод, но нечувствителен к чётному числу ошибок и порядку символов.
4. Циклический избыточный код (CRC): Мощный метод на основе полиномиальной арифметики. Обнаруживает одиночные, двойные и нечётные ошибки. Использует деление битовой последовательности на генераторный полином, остаток (CRC) передаётся с данными. Приёмник проверяет целостность, сравнивая остатки.

CRC: Вычисляется побитно или таблично (для ускорения). Реализуется аппаратно благодаря простоте и эффективности. Пример: для сообщения 1101011011 и полинома 10011 CRC = 1110, передаётся как 11010110111110.

Итог: CRC — наиболее надёжный и распространённый метод, обеспечивающий высокую вероятность обнаружения ошибок при относительной простоте реализации.

**Задание по варианту:**

Изображение выглядит как текст, бумага, рукописный текст, Бумажное изделие

Контент, сгенерированный ИИ, может содержать ошибки.Изображение выглядит как текст, рукописный текст, бумага, блокнот

Контент, сгенерированный ИИ, может содержать ошибки.

**2. Листинги программы**

#include <iostream>

#include <string>

#include <random>

using namespace std;

// 1. Функция вычисления CRC

string calculateCRC(const string& input, int length, const string& polynomial) {

string augmented = input + string(polynomial.length() - 1, '0'); // Дополняем нули

string remainder = augmented.substr(0, polynomial.length()); // Начальный остаток

for (int i = polynomial.length() - 1; i < augmented.length(); ++i) {

if (remainder[0] == '1') { // Если старший бит 1, выполняем XOR

for (int j = 0; j < polynomial.length(); ++j) {

remainder[j] = (remainder[j] == polynomial[j]) ? '0' : '1';

}

}

// Сдвиг влево и добавление следующего бита

remainder = remainder.substr(1) + (i + 1 < augmented.length() ? augmented[i + 1] : '0');

}

return remainder.substr(0, polynomial.length() - 1); // Возвращаем CRC

}

// 2. Функция генерации случайной последовательности

string generateRandomSequence(int length) {

string sequence;

random\_device rd;

mt19937 gen(rd());

uniform\_int\_distribution<> dis(0, 1);

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

sequence += (dis(gen) == 0) ? '0' : '1';

}

return sequence;

}

// 3. Функция проверки CRC

bool verifyCRC(const string& sequence, const string& polynomial) {

string crc = calculateCRC(sequence, sequence.length(), polynomial);

return crc == string(polynomial.length() - 1, '0'); // Проверяем, что CRC = 0

}

int main() {

setlocale(LC\_CTYPE, "Russian");

// Тест 1: Заданное сообщение

string message = "110101101";

string polynomial = "10011";

cout << "Сообщение: " << message << endl;

string crc = calculateCRC(message, message.length(), polynomial);

cout << "CRC: " << crc << endl;

string transmitted = message + crc;

cout << "Переданная последовательность: " << transmitted << endl;

bool isValid = verifyCRC(transmitted, polynomial);

cout << "CRC проверки: " << (isValid ? "0 (верно)" : "не 0 (ошибка)") << endl;

// Тест 2: Случайная последовательность

int k = 1000;

string randomSeq = generateRandomSequence(k);

cout << "\nСлучайная последовательность (первые 20 бит): " << randomSeq.substr(0, 20) << "..." << endl;

crc = calculateCRC(randomSeq, randomSeq.length(), polynomial);

cout << "CRC: " << crc << endl;

transmitted = randomSeq + crc;

isValid = verifyCRC(transmitted, polynomial);

cout << "CRC проверки: " << (isValid ? "0 (верно)" : "не 0 (ошибка)") << endl;

return 0;

}

Результат выполнения программы: Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Выводы: В ходе выполнения работы были реализованы функции для вычисления CRC и генерации случайных последовательностей. Функция calculateCRC корректно вычисляет остаток от деления сообщения на полином "10011" с использованием побитовой операции XOR. Проверка показала, что добавление CRC к исходной последовательности и последующее вычисление CRC итоговой последовательности даёт нулевой остаток, что подтверждает правильность алгоритма. Генерация случайной последовательности длиной 1000 бит также успешно протестирована. Методы просты в реализации и эффективны для обнаружения ошибок в передаваемых данных.