ГУАП

КАФЕДРА № 42

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

доцент		В. А. Кузнецов
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ № 5.1

БИНАРНЫЕ ОПЕРАЦИИ

по курсу:

ОСНОВЫ ПРОГРАММИРОВАНИЯ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ гр. № 4326		Г. С. Томчук
	подпись, дата	инициалы, фамилия

Санкт-Петербург 2024

СОДЕРЖАНИЕ

1 Постановка задачи

Задача: реализовать алгоритм на языке C/C++, выполняющий поставленную задачу. Вариант задания, пример входных и выходных данных представлен в таблице 1. Глобальные параметры использовать запрещено; допустимо использование дополнительных функций.

Обязательно использование только поразрядных операций для выполнения задания, математические операции с индексами и счетчиками разрешены.

Таблица 1 – Вариант

N	Текст задания	Вход	Выход
7	Реализовать функцию, которая обнуляет K заданных случайных бит, текущее значение которых равно 1, числа Int. Если в числе недостаточно единичных бит, то вывести 0. Значения обнуляемых позиций вывести в качестве отладочной информации. Нулевой позицией считается младший бит числа Int.	255 = 0000 0000 0000 0000 0000 0000 1111 1111, 3	61 = 0000 0000 0000 0000 0000 0000 0011 1101, 2,6,7

2 Схема алгоритма решения

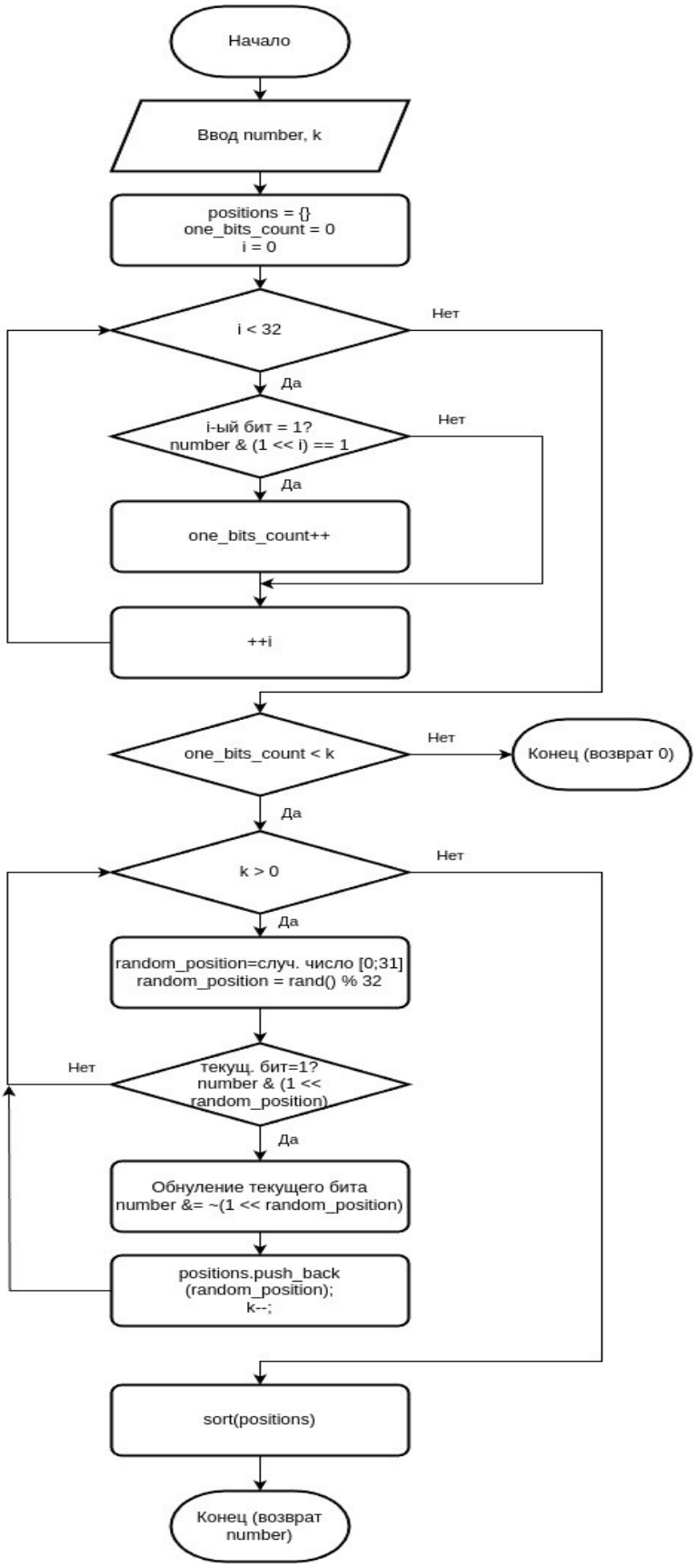

Рисунок 1 – Блок-схема алгоритма

3 Полное описание реализованной функции

Функция reset_random_bits обнуляет k случайных битов, равных 1, в числе number и сохраняет позиции обнуленных битов в векторе positions. Принимает следующие аргументы:

1. int number: Число, в котором будут обнуляться биты.
2. int k: Количество битов, которые нужно обнулить.
3. std::vector<int> &positions: Вектор для хранения позиций обнуленных битов.

Возвращает int — число после обнуления его k случайных битов. Если недостаточно единичных битов для обнуления, возвращает 0. Работа функции происходит следующим образом:

1. Подсчет единичных битов: циклом перебираем все 32 бита числа и с помощью единичной битовой маски подсчитываем, сколько из них равны 1.
2. Если единичных битов меньше, чем k, функция возвращает 0, сигнализируя, что обнуление невозможно.
3. Инициализируем генератор случайных чисел текущим временем.
4. Обнуление случайных битов: в цикле генерируются случайные позиции от 0 до 31 и с помощью той же маски проверяется, является ли бит на этой позиции единичным. Если да, бит обнуляется с помощью обратной единичной битовой маски, и позиция сохраняется в векторе positions. Уменьшаем k на 1 после каждого обнуления, пока k > 0.
5. Сортируем вектор позиций обнуленных битов для упорядоченного вывода.

Также в main происходит вывод отладочной информации: выводится число-результат, с помощью побитового сдвига и единичной битовой маски в цикле выводится двоичное представление числа-результата, а также вектор с позициями обнуленных битов.

4 Листинг программы

Листинг 1

```
#include <iostream>
#include <vector>
#include <algorithm>

int reset_random_bits(int number, int k, std::vector<int> &positions) {
    // Считаем количество единичных битов в числе
    int one_bits_count = 0;
    for (int i = 0; i < 32; ++i)
        if (number & (1 << i))
            one_bits_count++;
    if (one_bits_count < k) {
        return 0; // Единичных битов недостаточно для обнуления
    }

    srand(time(nullptr)); // Установка начала последовательности чисел,
генерируемой rand()

    // Обнуляем случайные биты
    while (k > 0) {
        int random_position = rand() % 32; // Генерация случайной позиции от 0
до 31
        if (number & (1 << random_position)) { // Проверка, является ли
текущий бит единицей
            number &= ~(1 << random_position); // Обнуление текущего бита
            positions.push_back(random_position); // Сохранение позиции
обнуленного бита
            k--;
        }
    }

    std::sort(positions.begin(), positions.end());

    return number;
}

int main() {
    int number;
    int k;

    std::cout << "Число: ";
    std::cin >> number;

    if (std::cin.fail()) {
        std::cerr << "Число слишком большое";
        return 1;
    }

    std::cout << "K: ";
    std::cin >> k;

    std::vector<int> positions;

    int result = reset_random_bits(number, k, positions);
```

Продолжение листинга 1

```
    // Выводим отладочную информацию
    if (result == 0) {
        std::cout << 0;
    } else {
        std::cout << result << " = ";
        int bit_counter = 0;
        for (int i = 31; i ≥ 0; --i) {
            if (bit_counter == 4) {
                std::cout << " ";
                bit_counter = 0;
            }
            std::cout << ((result >> i) & 1);
            bit_counter++;
        }

        std::cout << std::endl;

        for (size_t i = 0; i < positions.size(); ++i) {
            std::cout << positions[i];
            if (i < positions.size() - 1) std::cout << ", ";
        }
    }

    return 0;
}
```

5 Результаты тестирования программы

```
Число: 255
K: 3
91 = 0000 0000 0000 0000 0000 0000 0101 1011
2, 5, 7
Process finished with exit code 0
```

Рисунок 2

```
Число: 15
K: 2
9 = 0000 0000 0000 0000 0000 0000 0000 1001
1, 2
Process finished with exit code 0
```

Рисунок 3

```
Число: 15
К: 2
12 = 0000 0000 0000 0000 0000 0000 0000 1100
0, 1
Process finished with exit code 0
```

Рисунок 4

```
Число: 2147483647
К: 10
1337977049 = 0100 1111 1011 1111 1110 1000 1101 1001
1, 2, 5, 8, 9, 10, 12, 22, 28, 29
Process finished with exit code 0
```

Рисунок 5

```
Число: 2147483647
К: 10
365879037 = 0001 0101 1100 1110 1101 1110 1111 1101
1, 8, 13, 16, 20, 21, 25, 27, 29, 30
Process finished with exit code 0
```

Рисунок 6

```
Число: 2147483648
Число слишком большое
Process finished with exit code 1
```

Рисунок 7