МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«МИРЭА – Российский технологический университет»**
**РТУ МИРЭА**

Институт искусственного интеллекта
Базовая кафедра №252 – информационной безопасности

# ЛАБОРАТОРНАЯ РАБОТА №1 ПО ПРЕДМЕТУ «РАЗРУШАЮЩИЕ ПРОГРАММНЫЕ ВОЗДЕЙСТВИЯ»

**Студент группы ККСО-01-20** *Семин В.В.*

**Преподаватель** *Старший преподаватель*
*Трошков Вадим Евгеньевич*

Москва 2025

# ОГЛАВЛЕНИЕ

# ВВЕДЕНИЕ

По приведённому коду на языке СИ восстановить алгоритм фрагмента программы: составить блок-схему и описать неформально (словами). Например, в приведённом участке кода находится НОК трёх чисел и результат возводится в квадрат.

В случае, когда восстановленный алгоритм похож на известный, даже с небольшими оговорками, желательно это указать. Например, пузырьковый алгоритм, второй закон Ньютона, протокол тройного рукопожатия, игра шахматы и т. д.

В программах могут содержаться ошибки, обнаружение которых также будет плюсом (в первую очередь обратить внимание на ошибки безопасности).

# 1 TASK_STATIC1.CPP

Данная программа является реализацией игры «Морской бой».

Код в Листинге 1.1 является участком, где пользователь записывает координаты кораблей.

*Листинг 1.1 — Ввод координат кораблей пользователем*

```
printf("Input format... Begin from 4 to 1 \n\n\n");
for (int i = 0; i < 10; i++)
{
	for (int j = 0; j < counter; j++)
		scanf_s("%c", &start[i][j], sizeof(start));
	counter -= 2;
	ctr2 += 1;
	if (ctr2 == 2 || ctr2 == 4 || ctr2 == 5 || ctr2 >= 7)
		counter += 2;
}
```

Блок-схема данного кода представлена на Рисунке 1.1.

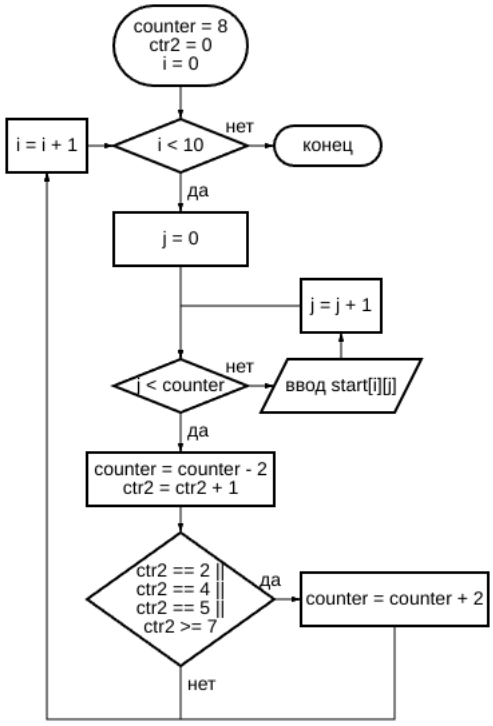


*Рисунок 1.1 — Ввод координат кораблей пользователем*

Ошибками являются наличие неиспользуемых ячеек в массиве start, что создаёт дополнительную поверхность атаки, а также отсутствие проверки ввода пользователя. Так как пользователь может ввести любую координату, это может привести к выходу за пределы массива в дальнейшем.

Функция *put* проверяет корректность размещения кораблей на поле компьютера. Её блок-схема представлена на Рисунке А.1 в Приложении А. Главной ошибкой в данное функции является отсутствие ключевого слова return в конце программы. Это может привести к неопределённому поведению программы в случае, если ни один из условных переходов не выполнился. Данный факт создаёт  дополнительную поверхность атаки для нарушителя.

Участок кода, ответственный за размещение кораблей на поле компьютера, представлен в Листинге 1.2.

*Листинг 1.2 — Генерация координат кораблей компьютера*

```
counter = 8;
ctr2 = 0;
int mrkr = 0;
for (int i = 0; i < 10; i++)
{
    for (int j = 0; j < counter; j += 2)
    {
                    mrkr = 0;
                    while (mrkr == 0) {
                        stcm[i][j] = rand() % 10;
                        stcm[i][j + 1] = rand() % 10;
                        mrkr = put(stcm[i][j], stcm[i][j + 1], a2, j, counter, stcm);
                    }
     }
    counter -= 2;
    ctr2 += 1;
    if (ctr2 != 1 && ctr2 != 3 && ctr2 != 6)
                    counter += 2;
}
```

Блок-схема данного участка представлена на Рисунке 1.2.

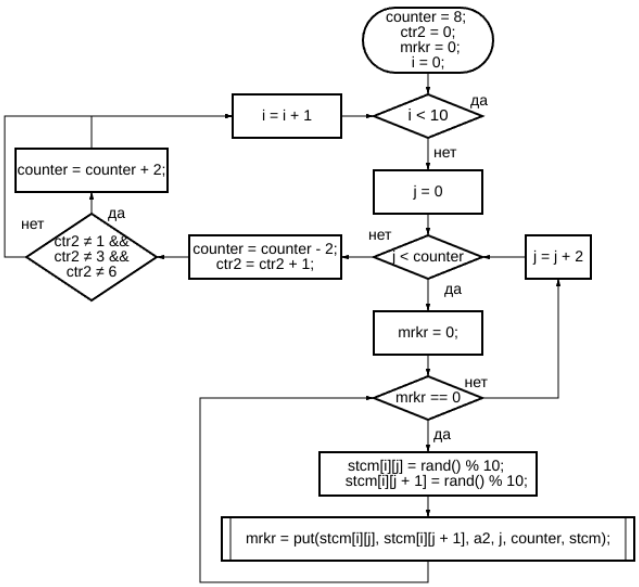


*Рисунок 1.2 — Генерация координат кораблей компьютера*

Функция *chc1* переводит координаты из формата, используемом при вводе пользователя, в формат, понятный самой программе. Её блок-схема представлена на Рисунке 1.3.

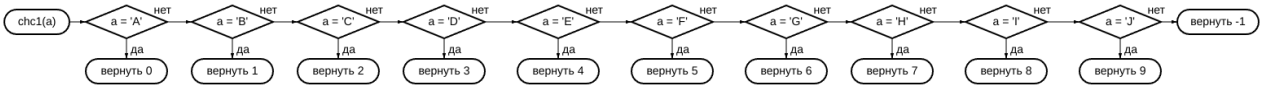


*Рисунок 1.3 — Функция chc1*

Следующий участок кода (Листинг 1.3) заполняет поля игрока согласно введённым им координатам.

*Листинг 1.3 — Заполнение поля игрока*

```
for (int i = 0; i < 10; i++)
{
	for (int j = 0; j < 7; j++)
	{
		int x = chc1(start[i][j]) + 1;
		int y = atoi(&start[i][j + 1]) + 1;
		if (j % 2 == 0 && start[i][j] != '\0')
			a1[y][x] = 'k';
		}
}
```

Блок-схема данного участка кода представлена на Рисунке 1.4.

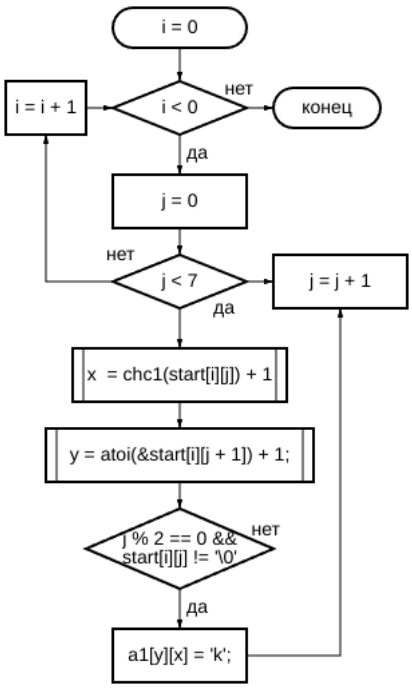


*Рисунок 1.4 — Заполнение поля игрока*

Ошибка данного участка кода заключается в том, что автор считывает неинициализированные части массива *start*, считая, что они были проинициализированы нулями. Хотя данный факт зависит от компилятора и операционной системы. Данный факт, помимо возможного неопределённого поведения программы, также создаёт поверхность атаки путём выхода за пределы массива.

Функция *kli* отвечает за обработку при попадании корабля. Её блок-схема представлена на Рисунке А.2 в Приложении А.

Функция *goin* отвечает за обработку хода игрока. Её блок-схема представлена на Рисунке А.3 в Приложении А.

Функция *gcm* отвечает за обработку хода компьютера. Её блок-схема представлена на Рисунке А.4 в Приложении А.

Функция *end* проверяет, закончилась ли игра. Её блок-схема представлена на Рисунке 1.5.

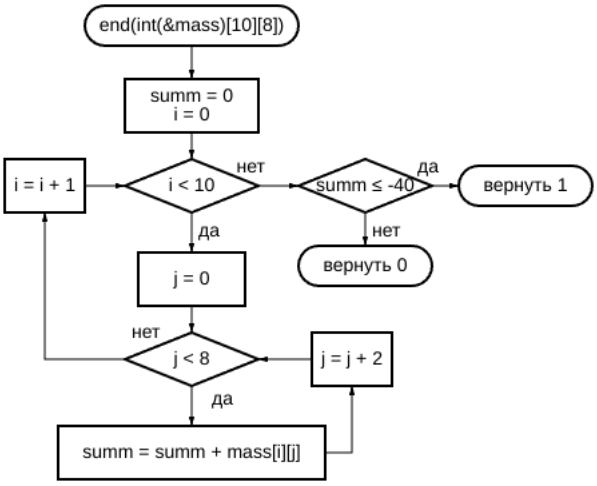


*Рисунок 1.5 — Функция end*

Код в Листинге 1.4 является основным циклом игры.

*Листинг 1.4 — Основной цикл игры*

```
int edg = 0;
while (edg != 1)
{
	goin(stcm, z2, stcmbc);
	edg = end(stcm);
	if (edg == 1)
		break;
	gcm(stbc1, a1, stbc);
	edg = end(stbc1);
}
```

Его блок-схема представлена на Рисунке 1.6.

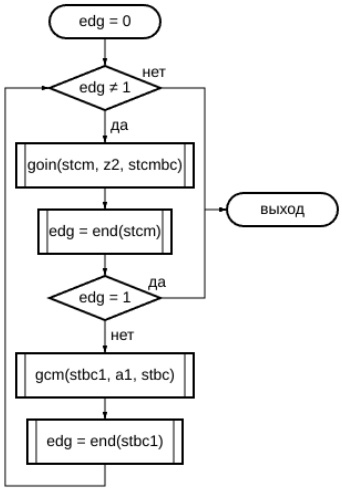


*Рисунок 1.6 — Основной цикл игры*

# 2 TASK_STATIC2.CPP

Данная программа является модификацией алгоритм блочного шифрования с длиной блока 64 бит, описанного в ГОСТ-34.12-2018, ГОСТ-34.12-2015 и ГОСТ 28147-89. Его общепринятое название - «Магма».

Определить это можно по следующим признакам.

1) Использование названия переменных *N1* и *N2* отсылает к терминам «Накопитель 1» и «Накопитель 2», использующихся при описании алгоритма в ГОСТ 28147-89. Причём, по строками *N2=N1; N1=CM2;* и *N1=N2; N2=CM2;* стразу можно понять, что используется сбалансированная сеть Фейстеля.
2) Битовые сдвиги в строке *left=R1<<stps; right=R1>>32-stps*.
3) Строка кода *if((x_cir>=0)&&(x_cir<=23)) CM1=N1+X[(x_cir%8)]; else CM1=N1+X[7-(x_cir%8)]*, которая является реализацией сложения раундового ключа с блоком по модулю $2^{32}$, причём в последних восьми раундах раундовые ключи накладываются в обратном порядке. Такое поведение (особенно использование арифметики по модулю $2^{32}$) является отличительной особенностью алгоритма «Магма».

Далее будут описаны особенности данной реализации.

1) Фиксированное количество блоков. Программа предполагает, что поданный на вход для шифрования файл будет содержать ровно 128000 блоков по 64 бит (определяется значением макроса rz).
2) Предполагается, что текст в файле будет будет представлен в виде hex-строки.
3) Пользователю дана возможность выбирать количество раундов и величину битового сдвига.

В данной реализации присутствуют следующие ошибки.

1) Массив подстановок *K* не инициализирован. Это означает, что он либо заполен нулями, либо мусором (зависит от компилятора и ОС). В первом

случае это является хоть и значительной, но не критической ошибкой, так как «Магма» обладает запасом прочности, позволяющим ей выдавать относительно хорошо распределённые блоки шифротекста. Второе приведёт к невозможности расшифровать полученный шифр текст, так как нельзя будет восстановить содержание подстановок на момент шифрования.

2) Объявление переменной *rz* после объявления макроса *rz*.
3) Массив раундовых ключей *X* не инициализирован. И снова, он либо нулевой, либо содержит мусор. Первое приведёт к лёгкому взлому шифрования, второе к невозможности расшифровать полученный шифр текст, так как нельзя будет восстановить содержание раундовый ключей на момент шифрования.
4) Использование функций *scanf* и *fscanf* может привести к переполнению буфера и легко эксплуатируется злоумышленниками.
5) Ввод пользователем значений количества итераций и величины битового сдвига, отличных от используемых в самом стандарте (32 и 11), приведёт к ослаблению шифрования.
6) Пропущена скобка, закрывающая блок цикла по блокам.
7) Использование режима шифрования ECB не допустимо на таком большом количестве блоков.
8) Переменная *i* используется без объявления.

Основной фрагмент программы (непосредственно, само шифрование) представлено в Листинге Б.1 з Приложения Б. Блок-схема этого фрагмента представлена на Рисунке Б.1 из Приложения Б.

# ЗАКЛЮЧЕНИЕ

В рамках данной лабораторной работы были проанализированы исходные коды двух программ. В результате анализа были восстановлены алгоритмы и составлены блок-схемы.

Первым алгоритмом оказалась реализация игры «Морской бой», а вторым модификация алгоритма «Магма».

Кроме этого были обнаружены ошибки реализации, которые могут привести к уязвимостям.

# ПРИЛОЖЕНИЕ А

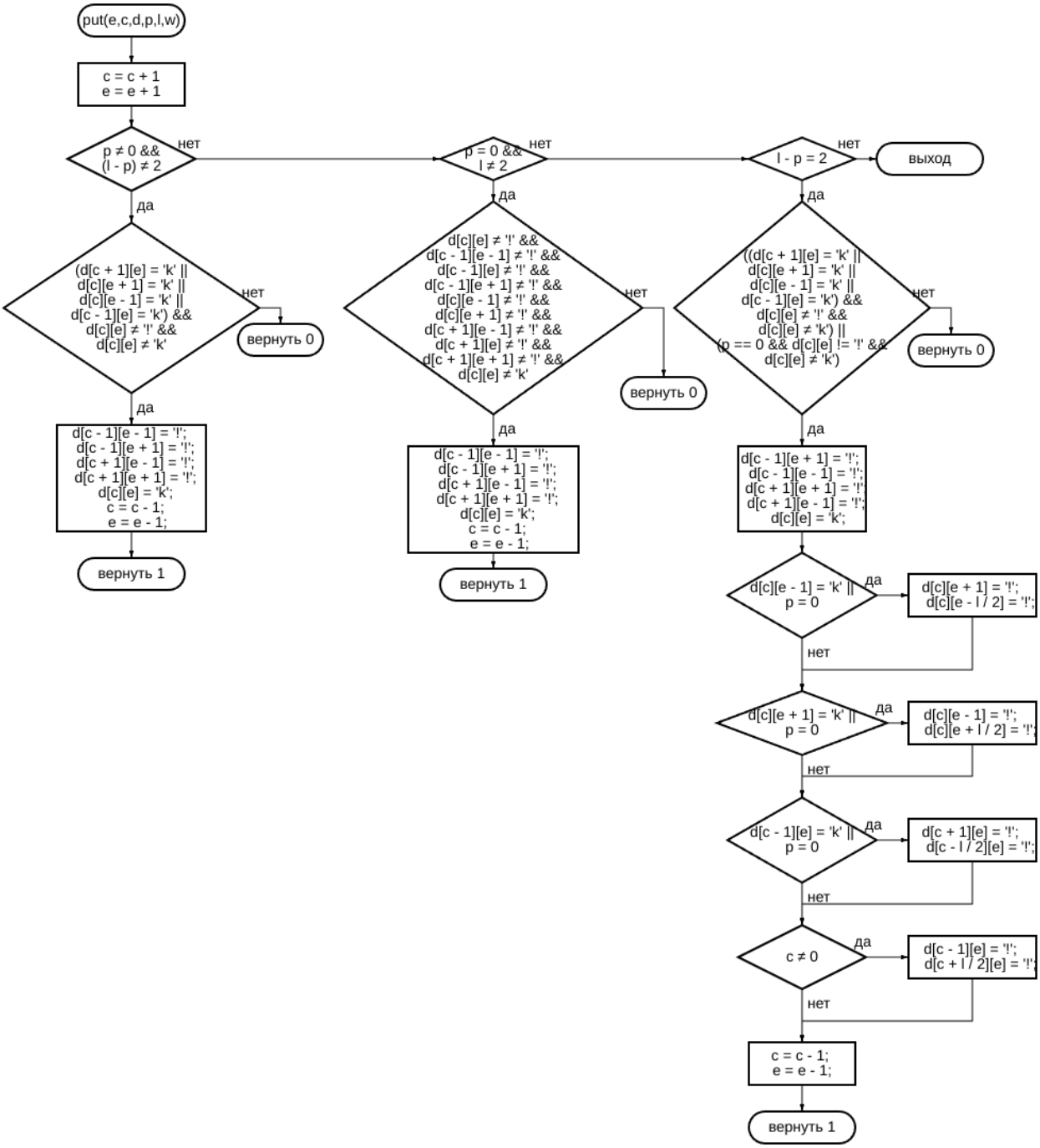


*Рисунок А.1 — Функция put*

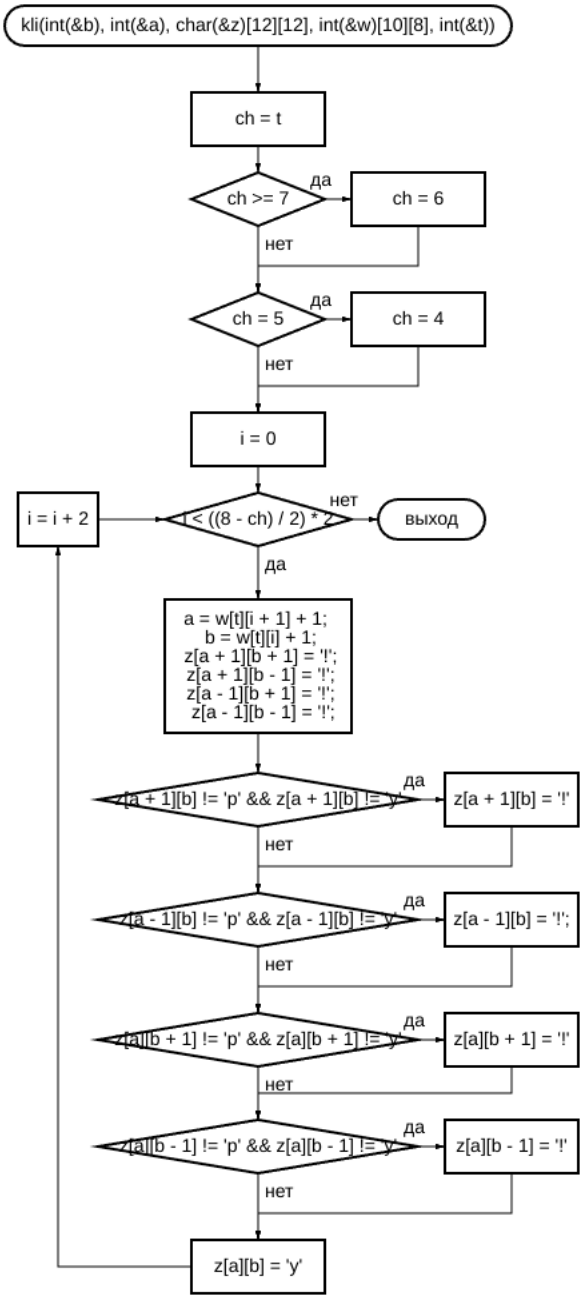


*Рисунок А.2 — Функция kli*

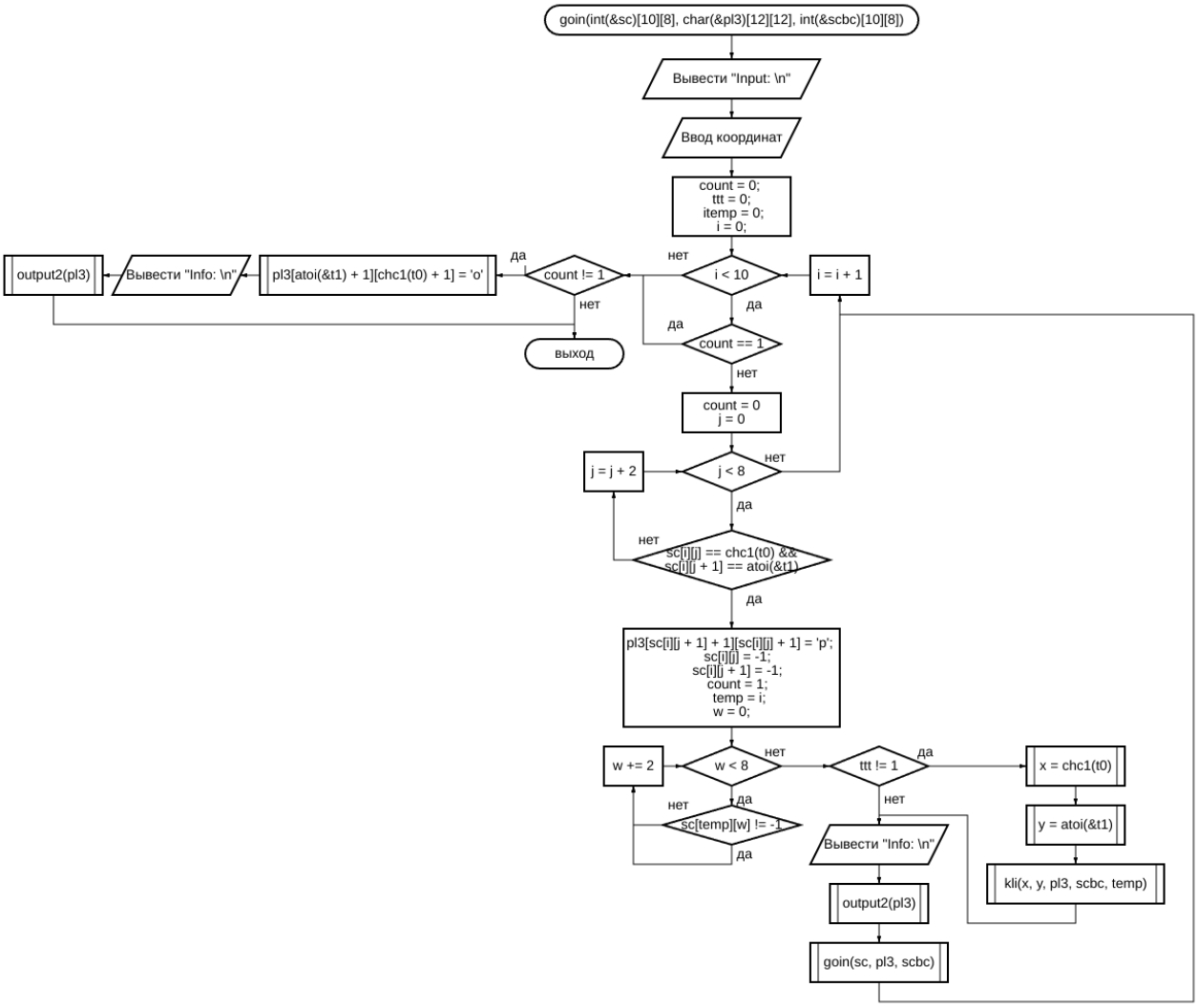


*Рисунок А.3 — Функция goin*

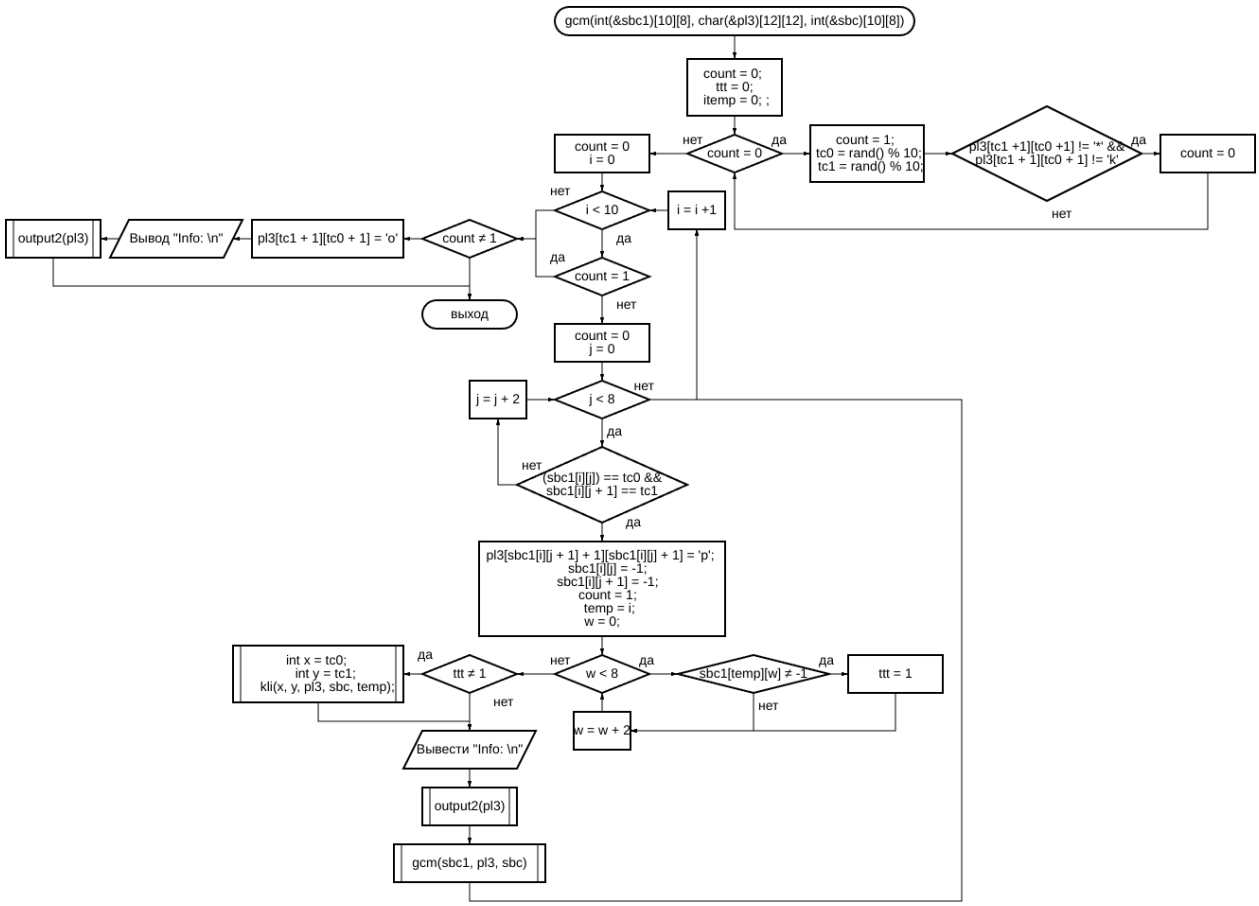


*Рисунок А.4 — Функция gcm*

# ПРИЛОЖЕНИЕ Б

*Листинг Б.1 — Реализация алгоритма шифрования «Магма»*

```
for (int ii=0; ii<rz; ii++)
 {
      N1=ch_mas[ii];
      N2=nch_mas[ii];

      int x_cir;
      for (x_cir=0; x_cir<itr; x_cir++)
      {
      if((x_cir>=0)&&(x_cir<=23)) CM1=N1+X[(x_cir%8)]; else CM1=N1+X[7-(x_cir%8)];

                for(i=0;i<8;i++)
                {
                        crcl=4*i;
                        CM1_using=CM1<<crcl; CM1_using=CM1_using>>28;
                        CM1_using=K[7-i][CM1_using];
                        R1=R1<<4; R1=R1+CM1_using;
                }
      left=R1<<stps; right=R1>>32-stps;
      rz=left+right;
      CM2=rz^N2;
      N2=N1; N1=CM2;
      }
      N1=N2; N2=CM2;
```

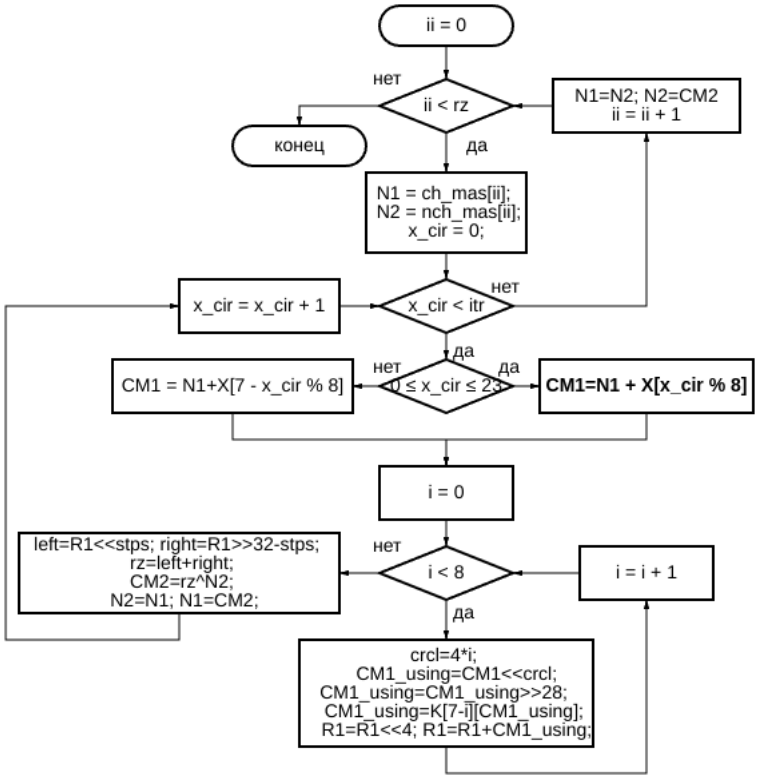


*Рисунок Б.1 — Реализация алгоритма шифрования «Магма»*