Критические ситуации в рекурсивных програмах

Рассмотрим на примере нашего кода

```cpp
#include <iostream>

using namespace std;

int nod(int a, int b) {
    if (b == 0) {
        return a;
    }
    return nod(b, a % b);
}

int main() {
    int x, y;

    cout << "Введите первое число: ";
    cin >> x;
    cout << "Введите второе число: ";
    cin >> y;

    int rez = nod(x, y);
    cout << "НОД(" << x << ", " << y << ") = " << rez << endl;

    return 0;
}
```

Критические ситуации

Переполнение стека

рекурсия всё равно создаёт риск переполнения стека при неправильных входных данных (например, числа Фибоначчи дают максимальную глубину, но для int безопасно).

Ввод отрицательных чисел

Обычно НОД — положительное число. Нужно брать abs() результата или входных аргументов.

Ввод нечисловых данных

cin не сможет извлечь число, установит failbit, переменная останется неинициализированной (мусор).

Вывод

Главная критическая ситуация — переполнение стека при большой глубине рекурсии.

Остальные критические ситуации рекурсивного подхода:

- Риск бесконечной рекурсии — при ошибочном изменении условия выхода или параметров рекурсивного вызова программа неизбежно упадёт из-за переполнения стека
- Зависимость от размера стека — код, работающий на ПК, может отказать на устройствах с малым стеком (встраиваемые системы)
- Затруднённая отладка — при переполнении стека теряется контекст вызовов, сложнее локализовать ошибку
- Избыточное потребление памяти — каждый рекурсивный вызов хранится в стеке, тогда как итеративное решение использует O(1) памяти