Análise de Erro: Resolvendo o "Wrong Answer" no Beecrowd 1069

Autor: Manus AI
Data: 13/08/2025
Assunto: Análise de erro de submissão em C, com foco em manipulação de buffer de entrada (stdin).

---

1. Resumo Executivo

Este documento técnico fornece uma análise detalhada do erro comum "Wrong Answer" encontrado ao resolver o problema **1069 - Diamantes e Areia** na plataforma Beecrowd (anteriormente URI Online Judge). Embora a lógica de resolução do problema, que envolve o uso de uma estrutura de dados de Pilha (Stack), possa estar correta, muitos desenvolvedores C encontram falhas devido a um manuseio inadequado do buffer de entrada padrão (`stdin`).

O erro frequentemente se manifesta quando a solução funciona para casos de teste simples, mas falha em um pequeno percentual dos casos de teste do juiz online, resultando em respostas como "Wrong Answer (5%)". A análise demonstra que a causa raiz não está na lógica do algoritmo, mas sim na interação problemática entre diferentes funções de leitura da biblioteca padrão de C, como `scanf()` e `fgets()` . A principal culpada é a permanência do caractere de nova linha (`\n`) no buffer de entrada, que corrompe as leituras subsequentes.

Este guia irá detalhar a causa do problema, ilustrar o comportamento incorreto com exemplos práticos e apresentar uma solução robusta e definitiva utilizando `fgets()` em conjunto com técnicas para limpar o buffer, garantindo que o programa se comporte de maneira previsível e correta em todos os casos de teste.

2. O Problema: Interação de Funções de Leitura e o Buffer de Entrada

O problema 1069 do Beecrowd exige a leitura de um número inteiro `N` (quantidade de casos de teste) seguido por `N` linhas de strings. Em C, a leitura de entrada é uma fonte comum de erros, especialmente quando diferentes funções de leitura são misturadas ou quando o buffer de entrada (`stdin`) não é gerenciado adequadamente.

2.1. O Comportamento de `scanf()`

A função `scanf()` é amplamente utilizada para ler dados formatados. Quando usada com o especificador `%d` para ler um inteiro, como em `scanf("%d", &n);` , ela lê os dígitos do número

e para quando encontra um caractere que não é um dígito (geralmente um espaço em branco, uma tabulação ou uma quebra de linha). O problema crucial é que `scanf()` **não consome o caractere de quebra de linha (`\n`)** que o usuário digita ao pressionar ENTER após inserir o número. Este `\n` permanece no buffer de entrada [1].

De forma similar, quando `scanf()` é usado com o especificador `%s` para ler uma string, como em `scanf("%s", str);`, ele lê caracteres não-espaço em branco até encontrar o próximo caractere de espaço em branco (espaço, tabulação, ou quebra de linha). Ele também **não consome o delimitador** (o espaço em branco ou `\n`) que causou sua parada. Além disso, `scanf("%s", ...)` é inerentemente inseguro, pois não permite especificar um tamanho máximo para a string, tornando-o vulnerável a *buffer overflows* se a entrada for maior que o array de destino [2].

2.2. O Problema do `\n` Residual

Considere o seguinte fluxo de execução típico para o problema 1069:

1. `scanf("%d", &n);`
   - Usuário digita `2` e ENTER.
   - `n` recebe `2`.
   - O caractere `\n` do ENTER permanece no `stdin`.
2. Primeira iteração do loop de casos de teste:
   - `char x[MAX_SIZE];`
   - `scanf("%s", x);`
   - Neste ponto, `scanf("%s", x);` encontra o `\n` que sobrou no buffer. Dependendo da implementação do compilador e do sistema operacional, ele pode interpretar isso como uma string vazia (lendo nada e deixando o `\n` lá) ou pular para a próxima linha, ou ainda, em alguns casos, ler um caractere inesperado se o buffer contiver lixo de memória.

Se `scanf("%s", x);` lê uma string vazia ou uma string diferente da esperada devido ao `\n` residual, a variável `x` conterá dados incorretos ou não conterá nada, levando a um processamento incorreto da lógica do diamante e, consequentemente, a um

resultado "Wrong Answer". O juiz online do Beecrowd testa o código com uma variedade de entradas, e é provável que um dos casos de teste exponha essa falha na leitura.

2.3. O Comportamento Inseguro de `gets()`

Embora o código original do usuário tenha sido corrigido para usar `scanf()`, é importante mencionar que a função `gets()` é ainda mais problemática. `gets()` lê uma linha inteira do `stdin` até encontrar uma quebra de linha ou o fim do arquivo, mas **não tem como saber o**

tamanho do buffer de destino. Isso significa que, se a linha lida for maior que o array fornecido, `gets()` continuará escrevendo dados além dos limites do array, sobrescrevendo outras partes da memória do programa. Este comportamento é conhecido como *buffer overflow* e é uma vulnerabilidade de segurança grave, razão pela qual `gets()` foi removido do padrão C11 e não deve ser utilizado em hipótese alguma [3].

2.4. Manifestação do Erro no Beecrowd

Quando o problema de leitura de entrada ocorre, o programa pode se comportar de maneiras imprevisíveis:

- **Leitura de String Vazia:** Em alguns casos, o `scanf("%s", ...)` pode ler uma string vazia devido ao `\n` residual, fazendo com que o loop de processamento da string não seja executado ou seja executado com dados inválidos.
- **Leitura de Lixo de Memória:** Se o buffer de entrada contiver dados inesperados (lixo de memória) após uma leitura mal sucedida, o `scanf("%s", ...)` pode tentar interpretar esses dados como parte da string, levando a um `strlen()` incorreto e a um processamento algorítmico completamente errado.
- **Dessincronização da Entrada:** O mais comum é que o `\n` residual consuma a primeira parte da próxima entrada esperada, dessincronizando o programa com os dados fornecidos pelo juiz online. Isso pode fazer com que o programa tente processar um número como se fosse uma string, ou vice-versa, resultando em valores numéricos absurdos (como o `430122451...` visto no erro) ou falhas de segmentação.

Esses cenários levam ao resultado "Wrong Answer" porque a saída do seu programa não corresponde à saída esperada para os casos de teste que ativam essa falha de leitura. O percentual de erro (e.g., 5%) indica que apenas uma fração dos casos de teste do Beecrowd é afetada por essa condição específica.

3. A Solução: Gerenciamento Robusto do Buffer de Entrada com `fgets()`

A solução para o problema de leitura de entrada em C reside no uso de funções que oferecem maior controle sobre o buffer de entrada e que são seguras contra *buffer overflows*. A função `fgets()` é a escolha recomendada para ler linhas de texto, e a combinação com `getchar()` ou `fgetc()` é essencial para limpar o buffer quando necessário.

3.1. Consumindo o `\n` Residual após `scanf()`

Após uma chamada a `scanf()` que lê um tipo de dado que não consome a quebra de linha (como `%d`), é crucial limpar o buffer de entrada antes de qualquer leitura subsequente de

string. A maneira mais simples de fazer isso é consumir o `\n` restante. Uma abordagem comum é usar `getchar()` [4]:

Plain Text

```
#include <stdio.h>

int main() {
    int numero;
    char linha[100];

    printf("Digite um numero: ");
    scanf("%d", &numero); // Lê o número, deixa o \n no buffer

    getchar(); // Consome o \n residual do buffer

    printf("Digite uma linha de texto: ");
    fgets(linha, sizeof(linha), stdin); // Agora lê a linha corretamente

    printf("Numero: %d\n", numero);
    printf("Linha: %s", linha); // fgets inclui o \n, então não adicione
outro no printf

    return 0;
}
```

Neste exemplo, `getchar()` lê e descarta o caractere `\n` que foi deixado no buffer por `scanf("%d", &numero);`. Isso garante que a próxima chamada a `fgets()` comece a ler do início da linha de texto esperada, e não do `\n` residual.

3.2. Leitura Segura de Strings com `fgets()`

Para ler as strings de entrada (as linhas da mina), `fgets()` é a função preferida. Ela é mais segura que `scanf("%s", ...)` e `gets()` porque permite especificar o tamanho máximo do buffer, prevenindo *buffer overflows*. Além disso, `fgets()` lê a quebra de linha (`\n`) se ela couber no buffer, o que é importante para o gerenciamento do buffer.

A sintaxe de `fgets()` é `char *fgets(char *str, int n, FILE *stream);` onde:

- `str` : Ponteiro para o array de caracteres onde a string lida será armazenada.
- `n` : O número máximo de caracteres a serem lidos (incluindo o `\n` e o caractere nulo `\0` de terminação). `fgets()` lerá no máximo `n-1` caracteres.
- `stream` : O fluxo de entrada de onde ler (geralmente `stdin`).

Como `fgets()` inclui o `\n` no buffer (se ele for lido), é comum removê-lo para evitar problemas em comparações de string ou formatação de saída. Isso pode ser feito

localizando o `\n` e substituindo-o por um caractere nulo (`\0`) [5].

Plain Text

```
#include <stdio.h>
#include <string.h>

// ... (código da pilha e outras funções) ...

int main() {
    int n;
    scanf("%d", &n);
    getchar(); // Consome o \n após a leitura de N

    for (int i = 0; i < n; i++) {
        // ... (inicialização da pilha e contador de diamantes) ...

        char x[MAX_SIZE];
        fgets(x, MAX_SIZE, stdin); // Lê a linha inteira
        x[strcspn(x, "\n")] = 0;   // Remove o \n do final da string, se
existir

        // ... (lógica de processamento da string) ...

        printf("%d\n", diamante);
    }
    return 0;
}
```

3.3. Código Corrigido para o Beecrowd 1069

Considerando as correções discutidas, a seção `main()` do código para o problema 1069 do Beecrowd deve ser estruturada da seguinte forma:

Plain Text

```
#include <stdio.h>
#include <string.h>

// Implementação da Pilha (struct e funções: iniciarPilha, isEmpty, isFull,
peek, size, push, pop)
// ... (Seu código da pilha aqui, sem os printf de erro dentro das funções
para evitar SAÍDA INESPERADA)

int main() {
    int n;
    scanf("%d", &n);
```

```
    getchar(); // Consome o \n que sobrou no buffer após a leitura de N

    for (int i = 0; i < n; i++) {
        pilha caixa; // Declara e inicializa a pilha para cada caso de teste
        iniciarPilha(&caixa);

        int diamante = 0; // Inicializa o contador de diamantes para cada
caso de teste

        char x[MAX_SIZE]; // Array para armazenar a string da mina
        fgets(x, MAX_SIZE, stdin); // Lê a linha inteira da mina
        x[strcspn(x, "\n")] = 0;   // Remove o \n do final da string, se
existir

        int comprimento = strlen(x); // Obtém o comprimento da string lida

        // Loop para percorrer cada caractere da string da mina
        for (int t = 0; t < comprimento; t++) {
            if (x[t] == '<') {
                // Se for '<', empilha
                push(&caixa, x[t]);
            } else if (x[t] == '>') {
                // Se for '>', verifica se há um '<' na pilha para formar um
par
                if (!isEmpty(&caixa)) {
                    pop(&caixa);      // Remove o '<' correspondente
                    diamante++;       // Incrementa o contador de diamantes
                }
            }
            // Caracteres '.' são ignorados automaticamente
        }
        printf("%d\n", diamante); // Imprime o total de diamantes para o
caso de teste, seguido de uma quebra de linha
    }

    return 0;
}
```

Observação Importante: É crucial que as funções da pilha (`push` , `pop` , `peek` , etc.) **não imprimam mensagens de erro** (como `"Erro: Pilha cheia!"` ou `"Erro: Pilha vazia!"`) quando submetidas a plataformas de juízes online como o Beecrowd. Qualquer saída que não seja a resposta exata esperada para o problema resultará em "Wrong Answer". As verificações de `isFull()` e `isEmpty()` devem ser usadas para controlar o fluxo do programa, não para imprimir mensagens de depuração.

4. Referências

[1] Cplusplus.com. `scanf`. Disponível em: https://www.cplusplus.com/reference/cstdio/scanf/. Acesso em: 13 ago. 2025.

[2] OWASP. `Buffer Overflow`. Disponível em: https://owasp.org/www-community/attacks/Buffer_overflow. Acesso em: 13 ago. 2025.

[3] Cplusplus.com. `gets`. Disponível em: https://www.cplusplus.com/reference/cstdio/gets/. Acesso em: 13 ago. 2025.

[4] GeeksforGeeks. `How to clear input buffer in C?`. Disponível em: https://www.geeksforgeeks.org/clearing-input-buffer-in-c-cpp/. Acesso em: 13 ago. 2025.

[5] Cplusplus.com. `fgets`. Disponível em: https://www.cplusplus.com/reference/cstdio/fgets/. Acesso em: 13 ago. 2025.