# ERROS DO PROGRAMA EXEMPLO

A seguir, os trechos contendo os erros e a nossa solução.

**Trecho #1:**

****

Variável contadora “cont\_p” não inicializada, o que acarreta no uso de lixo de memória por parte do programa ao incrementar.

**Solução:**



Faz-se a inicialização da variável, atribuindo-lhe 0.

**Trecho #2:**



A rotina, antes de produzir um item no buffer, precisa sobretudo verificar se o buffer se encontra cheio, e isso é feito através da primeira condicional if presente no início da função.

Existem dois problemas lógicos: o primeiro é que inicializamos nossa variável “wp” – responsável por apontar a posição do buffer em que a posição se encontra – justamente na última posição, que é um impedimento presente na comparação “***(rp + SIZEOFBUFFER – 1 != wp)***” para entrar na condicional, fazendo com que os produtores nunca calhem a produzir.

O segundo grande problema é que estamos diante do uso de um buffer **circular**, isso implica em afirmar que ao usarmos a posição do ponteiro como indicativo do quanto produzimos e consumimos, não temos o valor real de produção ou consumo, por sempre voltarmos a apontar para o começo do buffer, como visto no comando “***wp = start;***”.

**Solução:**



Dentre as soluções possíveis, utilizar dos contadores globais previamente declarados está entre uma das mais simples. Ao considerarmos que possuem o objetivo de controlar o número de itens produzidos e consumidos, bastamos estabelecer uma condição básica: se o número total de produzidos menos o número total de consumidos forem menores que o tamanho do buffer, significa que ele não está cheio, implementando assim, a condicional de maneira correta.

**#Trecho 3:**



Antes de consumirmos um item, precisamos verificar se ele ao menos existe, e isso é dado pela condicional if “***wp != rp***”, ou ao menos é essa a proposta. A condição em si não faz sentido lógico, visto que garantir que o endereço apontado pelo consumidor seja diferente que o endereço para onde aponta o produtor não implica em dizer que o buffer está vazio, exceto que ambos estivessem no começo do buffer e que ele **não fosse circular**.

O segundo problema está no incremento do endereço do consumidor. Se consumimos, avançamos com o endereço para o próximo item do buffer, mas a expressão “***rp+-;***” constitui um erro semântico e não realiza tal operação.

**Solução:**



Utilizamos novamente os contadores absolutos de quanto produzimos e quanto consumimos, e desta vez, garantimos que sempre o número de consumidos seja *menor* que ao total de produzidos, visto que assim impedimos que haja consumo quando a quantidade de produzidos seja *igual a 0*, além de evitar que se consuma mais do que foi produzido, devido ao contador de consumidos sempre parar quando este é *igual* ao de produzidos, impedindo também que seja maior. O segundo erro, apenas corrige-se a aritmética, incrementando a variável rp.

**Trecho #4:**

****

Encontramos alguns erros lógico-aritméticos, como a falta de inicialização em sum, e dois erros dentro da condicional if, um em cont\_p, que desejamos incrementar, e outro em sum, que não faz sentido subtrair se o próprio nome da variável significa soma. Por fim, temos um erro nos prints, com a especificação do formato errado e tratando o dado de maneira errada ao tentarmos printar um ponteiro void como um inteiro comum.

**Solução:**

****

Apenas atribuição e correção dos operadores aritméticos, além disso, corrigimos os prints arrumando a especificação do tipo de print para “%d” e utilizando de uma variável ponteira do tipo inteiro “***id***” para recebermos o id do thread passado por parâmetro pela função main, assim, printamos o conteúdo desse endereço.

**Trecho #5:**



Mesmos erros do trecho anterior no que diz respeito aos prints, porém um erro distinto aparece: a condição do while faz com que nunca entremos neste, visto que nossa variável “***cont\_c***” é inicializada com 0 e só é incrementada dentro do while, o qual tendo como condição “0 > NO\_OF\_ITERATIONS”, sempre resultará em falso.

**Solução:**

****

Inverte-se a condição do while para *menor*, visto que nosso propósito é que enquanto não se tenha consumido 100 (valor de NO\_OF\_ITERATIONS), continue tentando consumir. No mais, apenas as mesmas correções presentes no trecho anterior.