[Skip to content](https://chatgpt.com/#main)

**You said:**

em um codigo em c, eu peço para digitar um numero e um operação, exemplo "1 + 1", mas o usuario pode digitar com espaço, como removo o espaço do meu vetor de char?

**ChatGPT said:**

Para remover os espaços de um vetor de char em C, você pode percorrer a string e criar uma nova versão sem os espaços. Aqui está um exemplo simples de como fazer isso:

**Código em C para remover espaços de uma string:**

#include <stdio.h>

void remover\_espacos(char str[]) {

int i, j = 0;

// Percorre a string original

for (i = 0; str[i] != '\0'; i++) {

// Se o caractere não for um espaço, copiamos para a nova posição

if (str[i] != ' ') {

str[j++] = str[i];

}

}

// Finaliza a string sem espaços

str[j] = '\0';

}

int main() {

char str[100];

printf("Digite uma expressão (exemplo: 1 + 1): ");

fgets(str, sizeof(str), stdin); // Usamos fgets para ler a linha inteira

remover\_espacos(str); // Chama a função para remover os espaços

printf("Expressão sem espaços: %s\n", str);

return 0;

}

**Explicação do código:**

1. **Função remover\_espacos**: A função percorre cada caractere da string e, se não for um espaço, move ele para a nova posição. Ela elimina os espaços e faz a string continuar sem eles.
2. **Uso de fgets**: O fgets é utilizado para garantir que o usuário consiga digitar a linha inteira, incluindo espaços. O scanf não seria ideal nesse caso, pois ele não lida bem com espaços.
3. **Resultado**: O programa remove todos os espaços da string digitada pelo usuário e exibe a expressão sem espaços.

Esse código vai funcionar bem para remover espaços em qualquer parte da expressão digitada pelo usuário.

**You said:**

consegue fazer essa lógica em tabela para eu entender melhor?

**ChatGPT said:**

Claro! Vou usar uma tabela para ilustrar como a lógica funciona, passo a passo, enquanto percorremos a string para remover os espaços.

**Exemplo de Entrada:**

Imaginemos que o usuário digitou a expressão:

" 1 + 1 "

O objetivo é remover os espaços e deixar apenas a expressão:

"1+1"

**Tabela de Explicação:**

| **Índice** | **Caractere Original** | **Caractere Copiado** | **Nova String (Sem Espaços)** | **Índice de Inserção** |
| --- | --- | --- | --- | --- |
| 0 | ' ' | Não copiado | - | 0 |
| 1 | '1' | '1' | '1' | 1 |
| 2 | ' ' | Não copiado | '1' | 1 |
| 3 | '+' | '+' | '1+' | 2 |
| 4 | ' ' | Não copiado | '1+' | 2 |
| 5 | '1' | '1' | '1+1' | 3 |
| 6 | ' ' | Não copiado | '1+1' | 3 |
| 7 | '\0' | '\0' | '1+1' | 3 |

**Explicação:**

* No **índice 0**, encontramos um espaço (' '). Como queremos remover os espaços, não copiamos esse caractere para a nova string.
* No **índice 1**, encontramos o caractere '1'. Como não é um espaço, copiamos esse caractere para a nova string e avançamos para o próximo índice de inserção.
* No **índice 2**, encontramos outro espaço. Assim, novamente, não copiamos esse caractere.
* No **índice 3**, encontramos o caractere '+'. Como não é um espaço, copiamos para a nova string.
* O mesmo acontece para o **índice 4** (espaço) e o **índice 5** (número '1').
* Quando encontramos o final da string ('\0'), terminamos o processo.

**Resultado Final:**

A nova string, sem os espaços, será:

"1+1"

Espero que a tabela tenha ficado clara! Ela mostra como a string é construída de forma incremental, copiando apenas os caracteres que não são espaços. Isso ajuda a entender a lógica de como o programa processa a string.

**You said:**

entendi como funciona, se o valor for == ' ' ele volta pro for, o incrementador i altera o valor e o incrementador j só altera o valor quando entra no if, mudando o valor de posição quando != de ' '

**ChatGPT said:**

Exatamente! Você entendeu perfeitamente.

Aqui está um resumo mais detalhado do que acontece:

* O **índice i** percorre toda a string original. Ele passa por cada caractere, um de cada vez.
* O **índice j** é o que controla onde o próximo caractere "válido" (diferente de espaço) será colocado na nova versão da string. Ele só é incrementado quando um caractere diferente de ' ' (espaço) é encontrado, porque é quando copiamos um caractere para a nova posição.

**Processo passo a passo:**

1. **Inicialização**:
   * i começa em 0, percorrendo a string original.
   * j também começa em 0, mas só será incrementado quando encontrar um caractere válido (não espaço).
2. **Quando str[i] != ' '** (ou seja, o caractere não é um espaço):
   * Copia-se esse caractere para a posição str[j].
   * Em seguida, incrementa-se j para preparar para a próxima inserção.
3. **Quando str[i] == ' '** (ou seja, o caractere é um espaço):
   * O j **não é incrementado** porque não há necessidade de inserir o espaço. O laço continua para o próximo caractere (i é incrementado normalmente).
4. **Final da string**:
   * Quando o laço termina, str[j] é configurado como '\0' para garantir que a string resultante seja finalizada corretamente.

Esse comportamento faz com que a string resultante tenha todos os espaços removidos, sem perder os outros caracteres, e o array é mantido de forma contínua.

Você está certo ao pensar que o i vai avançar sempre, enquanto o j só avança quando necessário (quando encontramos um caractere válido). Isso garante que a string sem espaços seja construída de maneira eficiente.

Se você tiver mais dúvidas ou quiser ver algum outro exemplo, só avisar! 😄

Get smarter responses, upload files and images, and more.

Parte superior do formulário

Parte inferior do formulário

ChatGPT can make mistakes. Check important info.

**🔹 O que você precisa saber**

1. **Leitura de strings**
   * A expressão deve ser lida como uma **string** (char[] ou char \*).
   * Exemplo: "10+1/9"
2. **Tokenização (separar números e operadores)**
   * Precisa separar os **operandos** (números) e os **operadores** (+ - \* /).
   * Em C, pode usar strtok(), sscanf(), ou percorrer caractere por caractere.
3. **Precedência dos operadores**
   * Divisão e multiplicação têm prioridade sobre soma e subtração.
   * Exemplo: 10+1/9 → primeiro faz 1/9, depois soma com 10.
4. **Estrutura de dados para cálculo**
   * O mais usado é a **Notação Polonesa Reversa (RPN)**, também chamada de **Shunting Yard Algorithm** (do Dijkstra).
   * Você converte a expressão infixa (10+1/9) em pós-fixa (10 1 9 / +), e depois avalia usando **pilhas**.
5. **Operações matemáticas básicas**
   * Saber lidar com inteiros (int) e números decimais (float ou double).
   * Exemplo: em 1/9, se usar int, o resultado é 0.
   * Se usar double, resultado é 0.111....

**🔹 Caminho possível**

**1. Primeiro, faça uma versão simples:**

* Suponha que a expressão tem só **dois números e um operador**.
  + Exemplo: 10+5 ou 7/2.
  + Use sscanf(expr, "%d%c%d", &num1, &op, &num2) para extrair.

**2. Depois, avance:**

* Permitir múltiplos operadores.
* Criar uma função para respeitar precedência (\* e / antes de + e -).
* Aqui entra a ideia do **parsing com pilha**.

**3. Se quiser ir mais longe:**

* Implementar **Shunting Yard Algorithm** para converter para RPN.
* Avaliar a expressão pós-fixa com uma pilha.