Introdução a Técnicas de Programação

Ponteiros

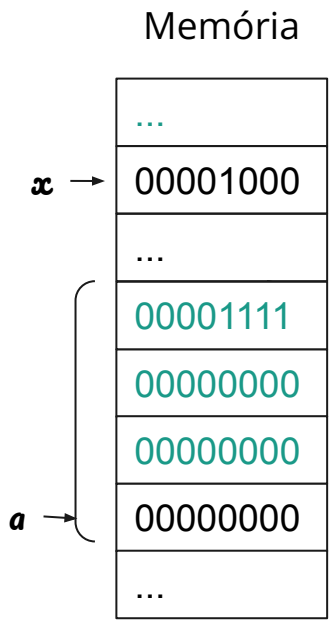
Prof. André Campos
DIMAp/UFRN

Ponteiros e endereços de memória

Como funciona a memória do computador

- Toda variável é associada a um endereço na memória
- Os tipos de variáveis definem quantos bytes elas ocupam, ou seja quantos endereços são usados

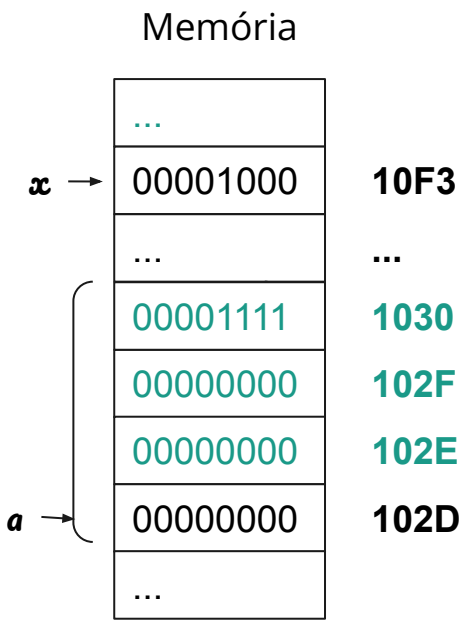
```
int a = 15; // 00000000000000000000000000001111
char x = 8; // 00001000
```



Como funciona a memória do computador

Espaços de 1 byte são “**endereçáveis**”.

- Os endereços são também valores numéricos.
- Normalmente, os endereços são representados em hexadecimal



Ponteiros

Se...

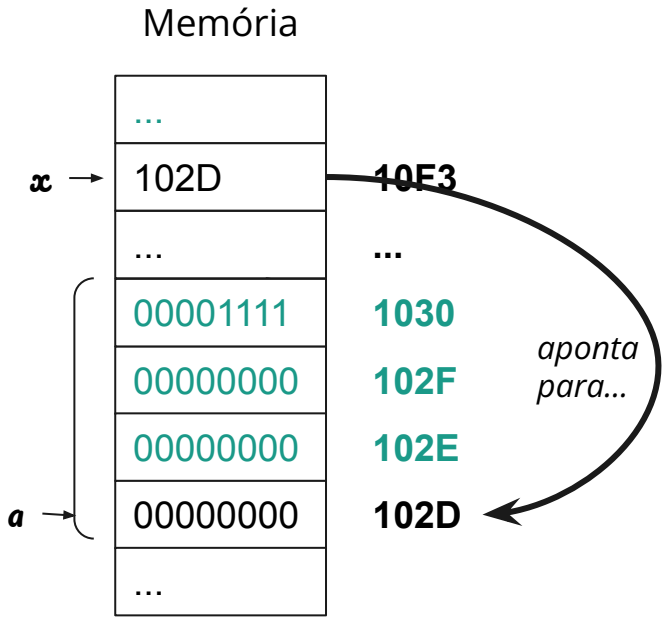
- um endereço é um valor numérico, e
- uma variável pode guardar um valor numérico

Então...

- uma variável pode armazenar um endereço

Ponteiros são variáveis especiais que guardam endereços de memória

"Apontam para um endereço"



a é do tipo int

x é do tipo "ponteiro para int"

Ponteiros (operadores)

Definição de um ponteiro

```
tipo *x;
```

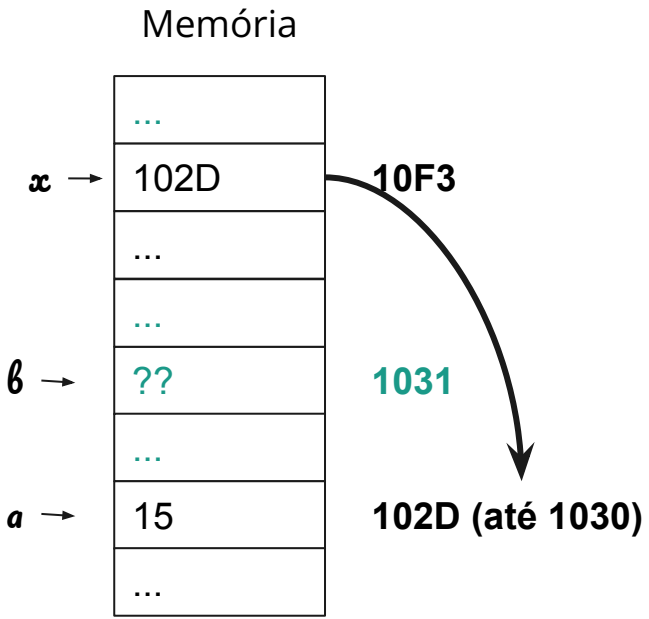
Operadores especiais

`&` → retorna o endereço de uma variável

`*` → retorna o conteúdo de um endereço

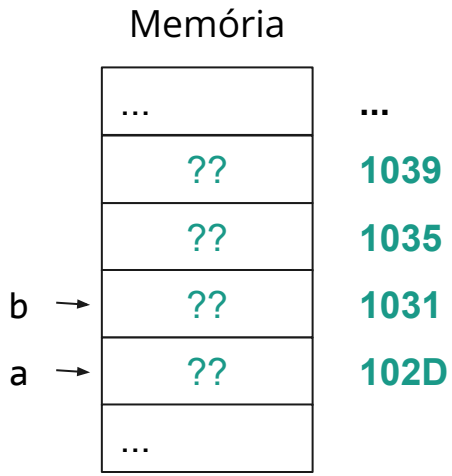
```
int a = 15;
int *x = &a;
int b = *x;
```

Atenção! Não confundir com o operador de multiplicação *(ex: `int c = 2 * *x;`)



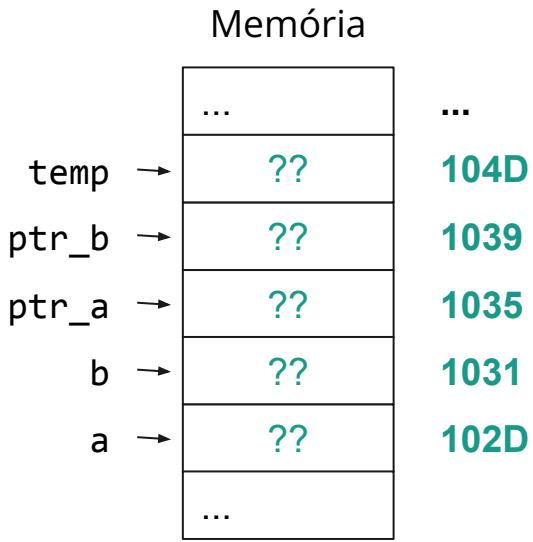
Exemplo 1

```
#include <iostream>
using namespace std;

int main() {
  int a = 10;  // variável inteira
  int *b = &a; // ponteiro para a variável inteira

  // altera o valor de a
  a = 20;
  cout << "Valor de b: " << *b << endl;

  // altera o valor de a através do ponteiro
  *b = 30;
  cout << "Valor de a: " << a << endl;

  return 0;
}
```



Exemplo 2

```
#include <iostream>
using namespace std;

int main() {
  int a = 10, b = 20;  // variáveis inteiras
  int *ptr_a = &a; // ponteiro para a variável a
  int *ptr_b = &b; // ponteiro para a variável b

  int temp = *ptr_a;
  *ptr_a = *ptr_b;
  *ptr_b = temp;

  cout << "Valor de a: " << a << endl;
  cout << "Valor de b: " << b << endl;

  return 0;
}
```



Ponteiro para “nenhum endereço”

Muitas vezes é necessário saber quando uma variável do tipo ponteiro foi inicializada, ou seja, quando ela está apontando para um **endereço válido**.

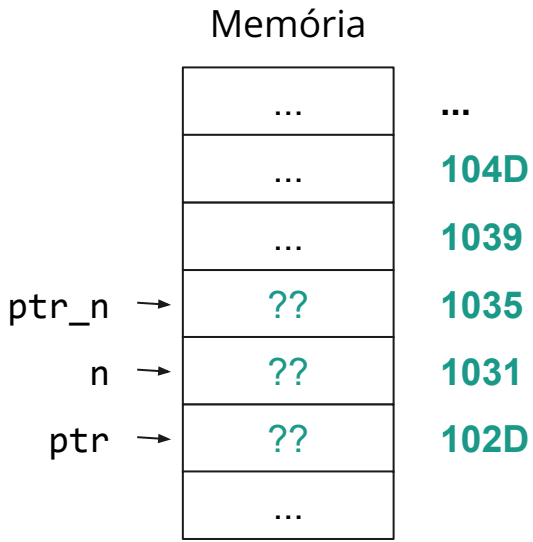
Como um endereço é um valor numérico qualquer, foi definido uma constante para indicar que o ponteiro não foi inicializado (está apontando pra nada): **nullptr**.

Dica: sempre que declarar uma variável ponteiro ou:

1. Atribua um endereço de variável para ela (ex: `int *ptr = &a;`), ou
2. Atribua o valor “nulo” (ex: `int *ptr = nullptr;`).

Exemplo

```
#include <iostream>
using namespace std;

int main() {
  int *ptr = nullptr; // ponteiro "nulo"

  int n = 0;
  int *ptr_n = &n; // ponteiro para n

  ptr = ptr_n; // ptr agora aponta para n

  *ptr = 10;
  cout << "Valor de n: " << n << endl;

  return 0;
}
```



Aplicações de ponteiros

- **Passagem de parâmetros por referência (passagem do endereço)**
  - Útil para alterar os valores de variáveis passadas como parâmetro
- **Variáveis vetores (arranjos) são variáveis ponteiros**
  - O bloco alocado para o vetor possui um endereço (ponteiro)
  - Variáveis que armazenam texto (strings) são vetores especiais, por isso são também ponteiros
- **Alocação dinâmica de variáveis**
  - Alocar um espaço em função, por exemplo, dos dados que o usuário fornece
- Estruturas de dados dinâmicas
  - Listas encadeadas, árvores, grafos ...
- Ponteiros de funções
  - Podemos tratar um função como uma variável qualquer, inclusive passado uma função como parâmetro para outra função

Ponteiros e linguagens de programação

Algumas linguagens evitam acesso direto à memória, apesar de tratarem internamente variáveis como referências (ponteiros), como em **Java** e **Javascript**.

Algumas linguagens permitem o acesso da memória em certas condições (tipos seguros), como em **C#**.

Outras linguagens permitem o acesso direto, como em **C**, **C++**, **Go** e **Rust**.

Ponteiros e arrays

Passagem de valores

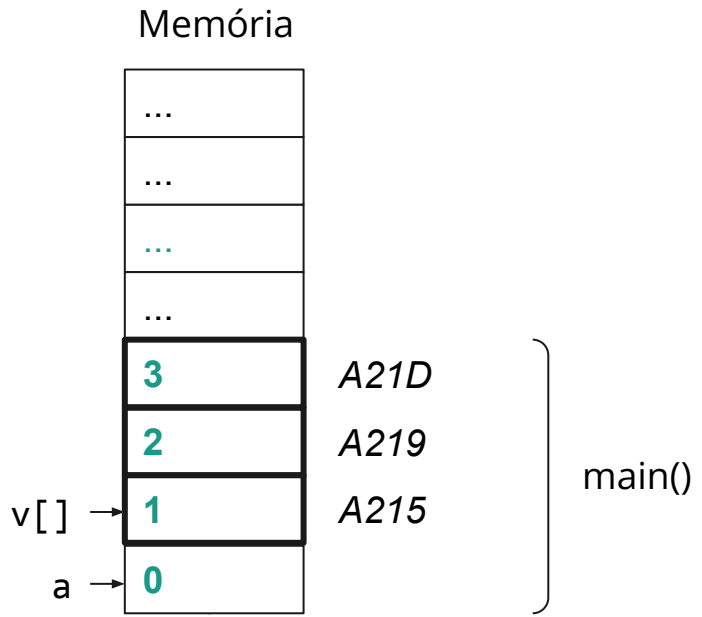
O que será impresso neste programa?

```
1  void func(int a, int v[3]) {
2    a = 5;
3    v[0] = 5;
4  }
5
6  int main() {
7    int a = 0;
8    int v[3] = { 1, 2, 3 };
9    func(a, v);
10   cout << a << " " << v[0];
11   return 0;
12 }
```

Variáveis arrays indicam blocos de memória

Uma variável array indica apenas o elemento inicial do bloco

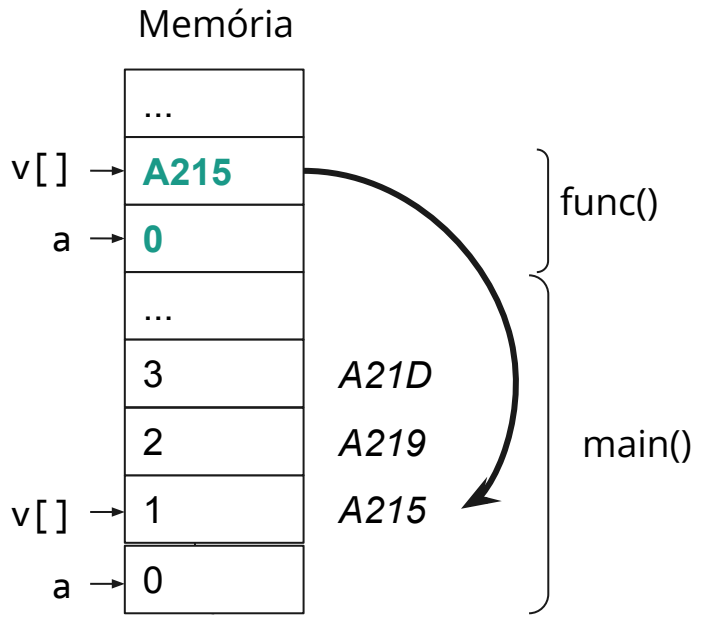
```
1  void func(int a, int v[3]) {
2    a = 5;
3    v[0] = 5;
4  }
5
6  int main() {
7    int a = 0;
8    int v[3] = { 1, 2, 3 };
9    func(a, v);
10   cout << a << " " << v[0];
11   return 0;
12 }
```



Arrays em parâmetros são convertidos em ponteiros

A variável do parâmetro guarda o endereço do elemento inicial do array

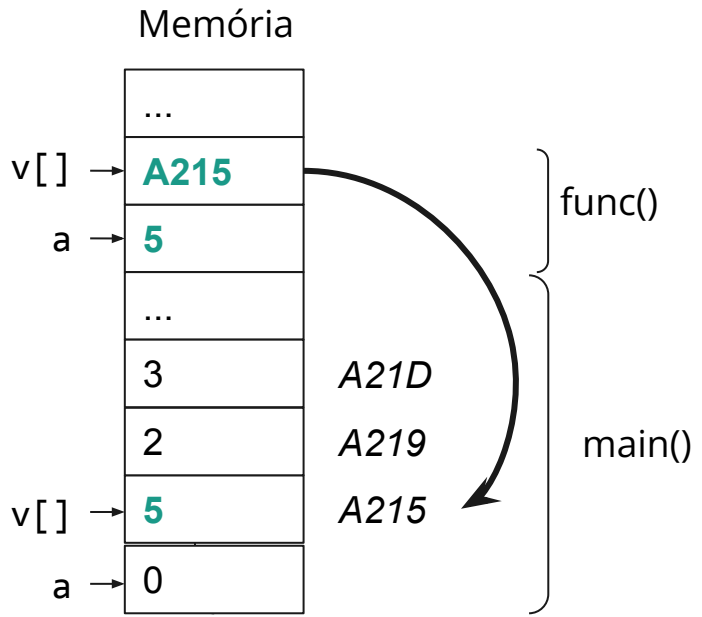
```
1  void func(int a, int v[3]) {
2    a = 5;
3    v[0] = 5;
4  }
5
6  int main() {
7    int a = 0;
8    int v[3] = { 1, 2, 3 };
9    func(a, v);
10   cout << a << " " << v[0];
11   return 0;
12 }
```



As alterações fora do escopo

As alterações a partir do ponteiro muda os valores fora do escopo

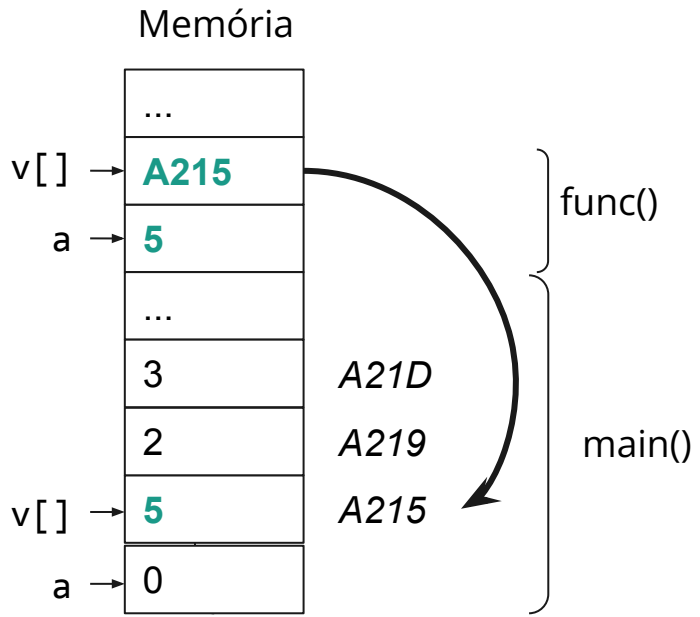
```
1  void func(int a, int v[3]) {
2    a = 5;
3    v[0] = 5;
4  }
5
6  int main() {
7    int a = 0;
8    int v[3] = { 1, 2, 3 };
9    func(a, v);
10   cout << a << " " << v[0];
11   return 0;
12 }
```



As alterações fora do escopo

As alterações a partir do ponteiro muda os valores fora do escopo

```
1  void func(int a, int v[3]) {
2    a = 5;
3    v[0] = 5;
4  }
5
6  int main() {
7    int a = 0;
8    int v[3] = { 1, 2, 3 };
9    func(a, v);
10   cout << a << " " << v[0];
11   return 0;
12 }
```



Aritmética de ponteiros

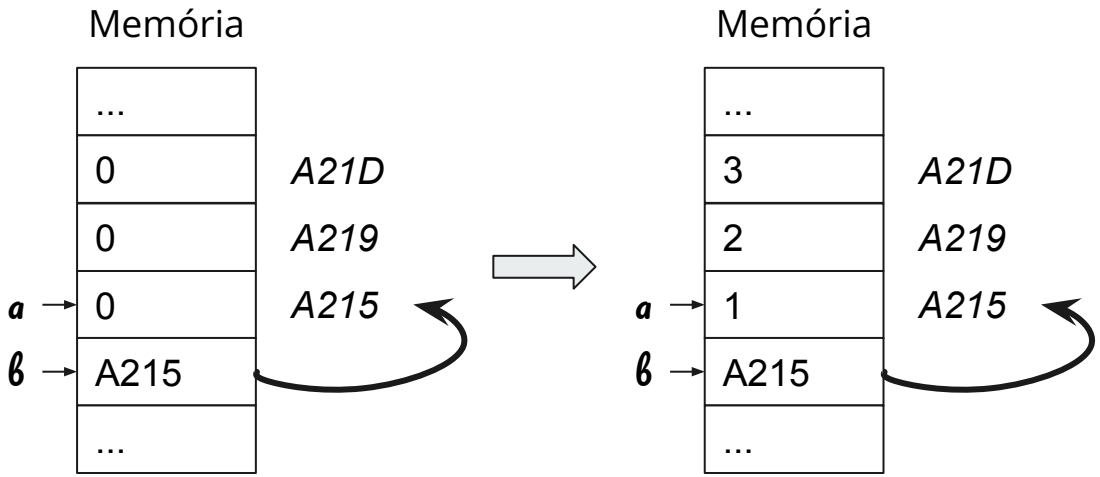
Um endereço, por ser um número, pode ser somado e subtraído

Somar um endereço de 1 resulta no “endereço seguinte”

- O “seguinte” depende do tipo de dado. O tipo `int` salta 4 valores porque usa 4 bytes.

Subtrair de 1 resulta no “endereço anterior”

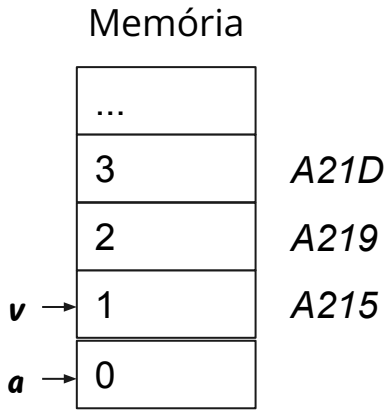
```c
int main() {
  int a = 0;
  int *b = &a;

  *b = 1;
  *(b + 1) = 2;
  *(b + 2) = 3;

  return 0;
}
```



Vetores e aritmética de ponteiros

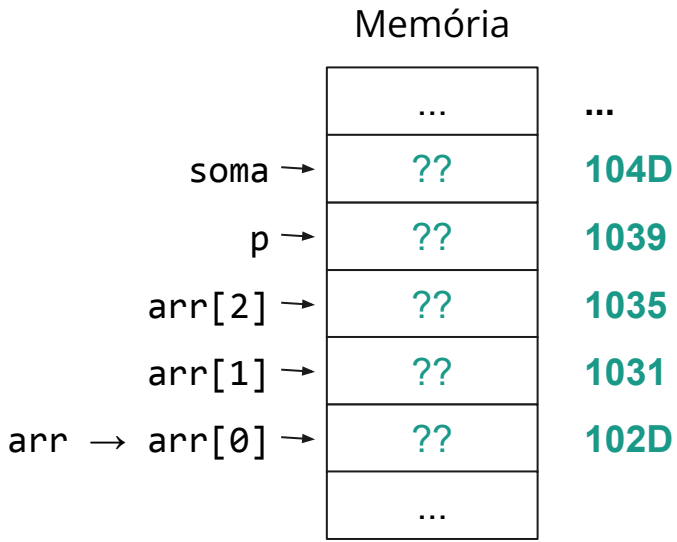
Vetores usam a soma e subtração de endereços para acessar seus dados
Acessar o valor de índice *i* de um vetor é acessar o conteúdo do endereço do vetor mais *i*. `a[i]` ⇐⇒ `*(a+i)`

```c
int main() {
  int a = 0;
  int v[3] = { 1, 2, 3 };

  *(v + 0) = 1;  // v[0] = 1;
  *(v + 1) = 2;  // v[1] = 2;
  *(v + 2) = 3;  // v[2] = 3;

  return 0;
}
```



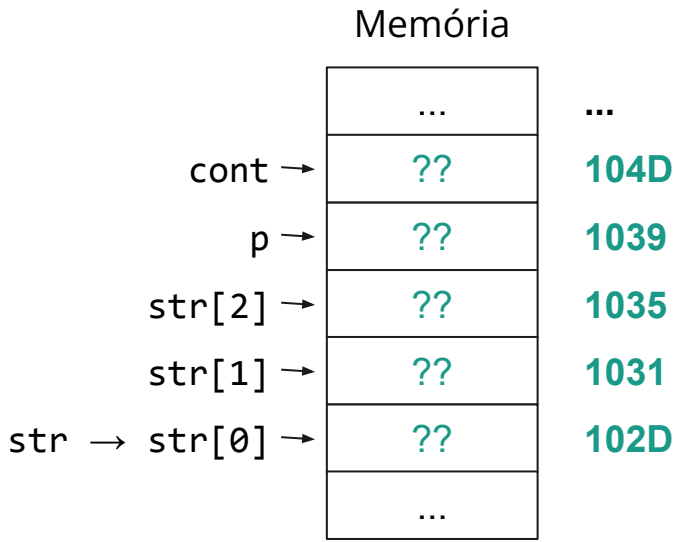
Exemplo 1

```
#include <iostream>
using namespace std;

int main() {
    int arr[] = { 1, 2, 3 };
    int *p = arr;
    int soma = 0;

    for (int i = 0; i < 3; i++) {
        soma += *p;
        p = p + 1; // ou p++
    }

    cout << "Soma: " << soma << endl;
    return 0;
}
```



Exemplo 2

```
#include <iostream>
using namespace std;

int main() {
    char str[] = "xy";
    char *p = str;
    int cont = 0;

    while (*p != '\0') {
        cont++;
        p++;
    }

    cout << "Tamanho: " << cont << endl;
    return 0;
}
```



Auto-avaliação

https://forms.gle/FLNNzYFNQUC2aiz27