

Algoritmos e Programação II

<https://evandro-crr.github.io/alg2>

C-String

Strings são sequências contínuas de caracteres.

- Na linguagem C, strings são arrays de `char` terminados com o caractere nulo (`'\0'`).
- Um literal de C-string retorna um `char[]` .

```
char str[] = "palavra";
```

0	1	2	3	4	5	6	7
'p'	'a'	'l'	'a'	'v'	'r'	'a'	'\0'

Por que estudar C-string se o C++ tem `std::string` ?

- Muitas bibliotecas de C++ usam C-strings.
- É comum usar bibliotecas de C em C++.

C-String com Array

- Podemos atribuir um literal de string a um array.
- O tamanho da string é a quantidade de caracteres antes do caractere nulo (`'\0'`).
- O tamanho do array deve ser, no mínimo, o tamanho da string+1.

```
char str[8] = "teste";
```

0	1	2	3	4	5	6	7
't'	'e'	's'	't'	'e'	'\0'	0	0

Lendo C-String com `cin`

```
const int TAMANHO = 21;  
char nome[TAMANHO];  
cin >> nome;
```

- O `cin` não sabe o tamanho do `char[]`.
- Se a string lida for maior do que 20 caracteres, o `cin` vai escrever além do array.

```
const int TAMANHO = 21;  
char nome[TAMANHO];  
cin.getline(nome, TAMANHO);
```

- O `cin.getline` garante que a leitura vai parar em 20 caracteres.
- Se tentar ler mais de 20 caracteres, um flag de erro será levantado no `cin`.

`cin.getline` e `cin.ignore`

```
const int tamanho = 21;
char buffer[tamanho];

cin.getline(buffer, tamanho);

if (cin.fail()) {
    cin.clear();
    cin.ignore(
        numeric_limits<streamsize>::max(),
        '\n'
    );
    cout << "O restante da "
         << "linha foi ignorado.\n";
}
```

- Caso tente ler mais elementos do que cabem no buffer, `cin.fail()` retornará `true`.
- Podemos usar `cin.ignore` para ignorar o restante da linha.
 - Por padrão, `cin.ignore` ignora 1 caractere.
- O `cin.clear` limpa o estado de erro.

Funções para char

Função	Descrição
<code>isalpha(char)</code>	Verdadeiro se for uma letra do alfabeto.
<code>isalnum(char)</code>	Verdadeiro se for uma letra do alfabeto ou um dígito.
<code>isdigit(char)</code>	Verdadeiro se for um dígito de 0 a 9.
<code>islower(char)</code>	Verdadeiro se for uma letra minúscula.
<code>isupper(char)</code>	Verdadeiro se for uma letra maiúscula.

Funções para char

Função	Descrição
<code>isprint(char)</code>	Verdadeiro se for um caractere imprimível.
<code>ispunct(char)</code>	Verdadeiro se for um caractere de pontuação.
<code>isspace(char)</code>	Verdadeiro se for um caractere de espaço em branco.
<code>toupper(char)</code>	Retorna o equivalente em maiúscula do argumento.
<code>tolower(char)</code>	Retorna o equivalente em minúscula do argumento.

Exemplo de Funções para char

```
char ch;
cout << "Digite um caractere: ";
cin >> ch;
if (isalpha(ch)) {
    cout << ch << " é uma letra do alfabeto.\n";
    if (islower(ch)) {
        cout << ch << " é uma letra minúscula.\n";
        cout << "Convertido para maiúscula: " << (char)toupper(ch) << "\n";
    } else if (isupper(ch)) {
        cout << ch << " é uma letra maiúscula.\n";
        cout << "Convertido para minúscula: " << (char)tolower(ch) << "\n";
    }
}
if (isdigit(ch))
    cout << ch << " é um dígito.\n";
if (isalnum(ch))
    cout << ch << " é um caractere alfanumérico.\n";
if (isprint(ch))
    cout << ch << " é um caractere imprimível.\n";
if (ispunct(ch))
    cout << ch << " é um caractere de pontuação.\n";
if (isspace(ch))
    cout << ch << " é um caractere de espaço em branco.\n";
```

Funções para C-String

size_t strlen(const char*)

Retorna o tamanho da string.

```
size_t strlen(const char* start) {  
    const char* end = start;  
    while (*end != '\\0')  
        ++end;  
    return end - start;  
}
```

Funções para C-String

char* strcat(char*, const char*)

Concatena duas strings.

```
char* strcat(char* dest, const char* src) {  
    char* ptr = dest + strlen(dest);  
  
    while (*src != '\0')  
        *ptr++ = *src++;  
  
    *ptr = '\0';  
  
    return dest;  
}
```

Funções para C-String

char* strcpy(char*, const char*)

Copia uma string.

```
char* strcpy(char* dest, const char* src) {  
    char* ptr = dest;  
  
    while ((*ptr++ = *src++) != '\0') {}  
  
    return dest;  
}
```

```
char* strcpy(char* dest, const char* src) {  
    char* ptr = dest;  
  
    do {  
        *ptr = *src;  
        ptr++;  
        src++;  
    } while (*src != '\0');  
  
    *ptr = '\0';  
  
    return dest;  
}
```

Funções para C-String

```
const char* strstr(const char*, const char*)
```

Procura o início de uma substring.

```
const char* strstr(const char* haystack, const char* needle) {  
    if (*needle == '\0') return haystack;  
    const char* h = haystack, *n;  
    while (*h != '\0') {  
        const char* h_start = h;  
        n = needle;  
        while (*h != '\0' && *n != '\0' && *h == *n) {  
            h++; n++;  
        }  
        if (*n == '\0') return h_start;  
        h = h_start + 1;  
    }  
    return nullptr;  
}
```

Funções para C-String

```
int strcmp(const char*, const char*)
```

Compara lexicograficamente duas strings.

```
int strcmp_custom(  
    const char* lhs,  
    const char* rhs  
) {  
    while (*lhs != '\0' && *lhs == *rhs) {  
        ++lhs;  
        ++rhs;  
    }  
  
    return *(unsigned char*)lhs  
        - *(unsigned char*)rhs;  
}
```

- Valor negativo se `lhs` aparecer antes de `rhs`.
- Zero se `lhs` e `rhs` forem iguais.
- Valor positivo se `lhs` aparecer depois de `rhs`.

Conversão Numérica

Função	Descrição
<code>atoi(char*)</code>	Converte a C-string para um número inteiro e retorna esse valor. Exemplo de Uso: <code>int num = atoi("4569");</code>
<code>atol(char*)</code>	Converte a C-string para um número inteiro longo e retorna esse valor. Exemplo de Uso: <code>long num = atol("4569");</code>
<code>atof(char*)</code>	Converte a C-string para um número float e retorna esse valor. Exemplo de Uso: <code>float num = atof("12.345");</code>

String em C++ `std::string`

- A classe `std::string` da biblioteca padrão do C++ permite a manipulação de strings de forma dinâmica e segura.
- Diferente das C-strings, o `std::string` abstrai a alocação de memória e manipulação de ponteiros.

Vantagens do `std::string`

- Não há necessidade de se preocupar com alocação ou liberação de memória.
- Métodos para comparação, concatenação, busca e modificação de strings.
- Pode-se obter uma C-string a partir de um `std::string` usando o método `.c_str()`.

Operadores de Comparação

```
string str1 = "banana";  
string str2 = "maçã";  
  
if (str1 < str2) {  
    cout << str1 << " vem antes de " << str2 << " no dicionário.\n";  
}  
  
if (str1 == "banana") {  
    cout << "As strings são iguais.\n";  
}
```

- `==` , `!=` , `<` , `<=` , `>` , `>=` são sobrecarregados para comparação lexicográfica de strings.

Operadores de Concatenação

Strings podem ser concatenadas facilmente com o operador `+`.

```
string primeiro_nome = "Evandro";  
string ultimo_nome = "Rosa";  
string nome = primeiro_nome + " " + ultimo_nome;  
  
cout << nome << '\n';
```

- O operador `+=` também é suportado para concatenar e atribuir:

```
string s = "Bom";  
s += " dia";  
cout << s << '\n'; // Saída: "Bom dia"
```

Acessando C-strings em `std::string`

A função `.c_str()` retorna um ponteiro para a C-string, permitindo compatibilidade com funções da linguagem C que exigem C-strings.

```
string nome = "Evandro";  
const char* c_string = nome.c_str();  
printf("Meu nome é %s\n", c_string);
```

Conversão de tipos em `std::string`

O C++ oferece funções para converter strings para números e vice-versa.

- `stoi(str)` : Converte uma string para `int` .
- `stof(str)` : Converte uma string para `float` .
- `stod(str)` : Converte uma string para `double` .
- `to_string(num)` : Converte um número para `std::string` .

```
string numero_str = "123";  
int numero = stoi(numero_str);  
cout << "Número + 1: " << numero + 1 << '\n'; // Saída: 124  
float valor = 45.67;  
string valor_str = to_string(valor);  
cout << "Valor em string: " << valor_str << '\n'; // Saída: "45.670000"
```

Algoritmos e Programação II

<https://evandro-crr.github.io/alg2>