

Strings

Video desta aula

Uma string é uma sequência de caracteres que permite representar nomes, endereços e outras informações textuais.

Declaração

Em C, strings são implementadas como vetores de caracteres (tipo `char`) terminados pelo caractere especial `'\0'` (caractere cujo código numérico é zero). Esse caractere terminal é considerado no tamanho do vetor. As aspas duplas (“...”) são usadas para declarar strings constantes.

Exemplos:

```
// declara string variável com até 99 caracteres (mais o \0)
char nome[100] ;

// declara string constante (com o \0 no final)
char *profissao = "estudante" ;

// declara ponteiro para uma string (não aloca espaço para ela)
char *endereco ;

// ou
char endereco[] ;

// ponteiro aponta para string constante
endereco = "Rua da Batata, 1000" ;
```

Deve-se observar que uma string é um **vetor de caracteres**, portanto as duas declarações abaixo são equivalentes:

```
char codigo[] = "XT07A" ;
char codigo[] = { 'X', 'T', '0', '7', 'A', '\0' } ;
```

A visão da string como vetor permite o acesso aos seus caracteres individuais. O código abaixo converte uma string em maiúsculas, usando os códigos ASCII dos caracteres:

```
i = 0 ;
while (str[i] != '\0') // varre todos os caracteres
{
    if (str[i] >= 'a' && str[i] <= 'z') // se for letra minúscula
        str[i] -= 32 ; // converte em letra maiúscula
    i++ ;
}

// ou, usando strlen()
for (i = 0; i < strlen (str); i++) // varre todos os caracteres
    if (str[i] >= 'a' && str[i] <= 'z') // se for letra minúscula
        str[i] -= 32 ; // converte em letra maiúscula
```



Strings declaradas usando `char[]` são normalmente usadas para armazenar texto codificado no padrão ASCII (texto simples, sem letras acentuadas). Para armazenar textos com caracteres não-ASCII (acentos, cedilha emojis, etc) devem ser usadas outras [codificações](#), como UTF8, e [strings multibyte](#).

Leitura e escrita

A escrita de strings pode ser feita com `printf` (usando o formato `%s` ou `%NNs`), `puts` ou ainda `putchar` (para escrever caractere por caractere):

```
char nome[] = "Homer Simpson" ;

printf ("Nome: %s\n", nome) ;           // escrita com printf

puts (nome) ;                           // escrita com puts

for (i = 0; i < strlen(nome); i++)     // escrita com putchar
    putchar (nome[i]) ;
```

Por sua vez, a leitura de strings pode ser feita usando a função `scanf`:

```
#define SIZE 100

char nome[SIZE+1] ; // não esquecer do '\0' no final da string

printf ("Digite seu nome: ") ;

// lê até encontrar espaço, tabulação, nova linha ou fim de arquivo
scanf ("%s", nome) ;

// idem, no máximo 20 caracteres
scanf ("%20s", nome) ;

// lê somente letras e dígitos (até encontrar outro caractere)
scanf ("%[A-Za-z0-9]", nome);

// lê até encontrar um fim de linha (\n), ou seja
// lê enquanto não encontrar um caractere '\n'
scanf ("%[^\\n]", nome);
getchar() ; // para ler o "\\n" no fim da linha
```



Observe que a leitura de uma string deve ser feita em uma variável com **espaço suficiente** para recebê-la (incluindo o `'\0'`), para não gerar um estouro de buffer ([buffer overflow](#)).

Pode-se também usar a função `fgets`:

```
// lê da entrada padrão até encontrar \n ou SIZE caracteres
fgets (nome, SIZE, stdin) ;

// a string lida por fgets pode incluir o \n do fim de linha,
// se ele foi encontrado; ele pode ser retirado assim:
```

```
nome[strlen (nome, "\n")] = '\0' ;
```

Para mais informações sobre as funções acima, deve ser consultada a respectiva página de manual Unix.



Existe uma função de leitura `gets()` que não limita o número de bytes lidos e pode provocar **estouro de buffer** e problemas de segurança, por isso **não deve ser usada!** Use a função `fgets()` em seu lugar.

Manipulação

A manipulação de strings é geralmente efetuada através de funções disponíveis na biblioteca padrão C, que podem ser acessadas através dos arquivos de cabeçalho `string.h` e `strings.h`.

Algumas das funções mais usuais são:

função	operação realizada
<code>int strlen (s)</code>	informa o número de caracteres da string <code>s</code> (sem considerar o <code>'\0'</code> no final)
<code>char * strcpy (b, a)</code>	copia a string <code>a</code> no local indicado por <code>b</code> ; a área de memória de destino deve ter sido previamente alocada (como variável normal ou dinâmica)
<code>char * strdup (s)</code>	Aloca uma área de memória dinâmica, copia a string <code>s</code> nela e devolve um ponteiro para a mesma
<code>char * strncpy (b, a, n)</code>	Copia <code>n</code> caracteres da string <code>a</code> no local indicado por <code>b</code>
<code>int strcmp (a, b)</code>	Compara as duas strings indicadas, retornando 0 se forem iguais, um valor negativo se <code>a < b</code> e um valor positivo se <code>a > b</code> , considerando a ordem alfabética
<code>int strncmp (a, b, n)</code>	Idem, mas só considera os <code>n</code> primeiros caracteres
<code>char * strcat (a, b)</code>	Concatena a string <code>b</code> ao final da string <code>a</code> (deve haver espaço disponível previamente alocado)
<code>char * strncat (a, b, n)</code>	Idem, mas só concatena os <code>n</code> primeiros caracteres
<code>char * strchr (s, c)</code>	Retorna um ponteiro para a primeira ocorrência do caractere <code>c</code> na string <code>s</code> , ou NULL se não encontrar
<code>char * strrchr (s, c)</code>	Idem, mas retorna um ponteiro para a última ocorrência do caractere

Várias outras funções para manipulação de strings estão disponíveis na [página de manual](#) (comando `man string`);

Exercícios

Escrever programas em C para:

1. Ler uma string da entrada padrão e escrevê-la na saída padrão ao contrário (do final para o início), de forma similar ao comando `rev` do *shell* UNIX.
2. Calcular o tamanho de uma string (sem usar `strlen`).
3. Converter as letras de uma string em minúsculas (dica: estude a estrutura da tabela ASCII antes de implementar).
4. Ler linhas da entrada padrão e escrevê-las na saída padrão em ordem alfabética crescente, de forma similar ao comando `sort` do *shell* UNIX.
5. Remover de uma string os caracteres que não sejam letras, números ou espaço, sem usar string auxiliar.
6. Remover de uma string caracteres repetidos em sequência (`rr`, `ss`, `ee`, etc), sem usar string auxiliar.

7. Colocar entre colchetes ([]) os caracteres de uma string que não sejam letras, números ou espaço; as alterações devem ser feitas na própria string, sem usar string auxiliar.
8. Escrever uma função `int busca(agulha, palheiro)`, que busca a string `agulha` dentro da string `palheiro`, sem usar funções prontas da biblioteca C. A função deve retornar o índice onde `agulha` começa em `palheiro`, -1 se não for encontrada ou -2 em caso de erro (uma ou ambas as strings são nulas).
9. Escrever sua própria versão das funções de manipulação de strings `strlen`, `strcpy` e `strcat`. Depois, comparar o desempenho de sua implementação em relação às funções originais da LibC (sugestão: meça o tempo necessário para ativar cada função um milhão de vezes).
10. Escrever uma função `palindromo(s)` que testa **palíndromos**: ela recebe uma string `s` de caracteres sem acentos e retorna 1 se a string é um palíndromo ou 0 senão. Acentos, espaços em branco e maiúsculas/minúsculas devem ser ignorados. Exemplos de palíndromos:
 - A cara rajada da jararaca
 - O poeta ama até o pó
 - Socorram-me, subi no ônibus em Marrocos!

From:

<https://wiki.inf.ufpr.br/maziero/> - **Prof. Carlos Maziero**

Permanent link:

<https://wiki.inf.ufpr.br/maziero/doku.php?id=c:strings>

Last update: **2023/08/24 18:28**

