

Strings

Representação de strings

Prof. Edson Alves - UnB/FGA

2018

1. Strings em C
2. Strings em C++
3. Strings em Python
4. Entrada e saída de string em console

Strings em C

Strings na linguagem C

- Em C, uma string é implementada como um *array* de caracteres terminado em zero (`'\0'`)
- Esta implementação é a mais sintética o possível em termos de memória: é reservado espaço apenas para armazenar os caracteres da string mais o terminador `'\0'`
- Em contrapartida, a ausência deste marcador pode levar a execução errônea de várias das funções que manipulam strings
- Além disso, rotinas simples, como determinar o tamanho da string, passa a ter complexidade $O(n)$, contrastando com as implementações que utilizam memória adicional e que podem retornar o tamanho em $O(1)$

Declaração e inicialização de strings em C

- Uma string pode ser declarada ou inicializada em C conforme os exemplos abaixo

```
char s1[101];           // Declaração da string s1
char s2[] = "Test";     // Inicialização da strings s2
```

- A strings s1, não inicializada, comporta até 100 caracteres (e o terminador '\0')
- A string s2, inicializada com o valor "Test", não exige que seja informado o número de caracteres e nem o terminador (o compilador completa tais informações automaticamente)
- Importante notar que, devido a aritmética de ponteiros da linguagem, as strings em C tem como primeiro elemento indexado em zero, não em um
- Assim, s[2] representa o terceiro, e não o segundo, elemento da string

Bibliotecas para manipulação de strings

- A biblioteca padrão do C oferece o *header* `string.h`, onde são declaradas várias funções para a manipulação de strings
- O *header* `stdlib.h` traz funções para conversão de strings para valores numéricos
- Ele também define três funções que permitem manipular a memória (`memcmp()`, `memset()`, `memcpy()`), através de comparações, atribuições e cópia, respectivamente, as quais são úteis para trabalhar com strings
- Outro arquivo útil para a manipulação de strings em C é o `ctype.h`, onde são definidas funções para a manipulação de caracteres

Exemplo de uso dos arquivos string.h e stdlib.h

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 int main()
6 {
7     char a[50] = "Test";
8     char b[] = "TEP";
9     char *p;
10
11     double d;
12     int i;
13     long long ll;
14
15     // Tamanho da string
16     int s = strlen(a);      // s = 4, zero terminador não é computado
17     s = strlen(b);          // s = 3
18
19     // Comparação
20     s = strcmp(a, b);        // s > 0, "Test" sucede "TEP" na ordem
21                             // lexicográfica
```

Exemplo de uso dos arquivos string.h e stdlib.h

```
22     s = strcmp(b, a);           // s < 0, "TEP" precede "Test" na ordem
23                                 // lexicográfica
24
25     s = strncmp(a, b, 1);       // s = 0, as strings são iguais no primeiro
26                                 // caractere
27     s = strncmp(a, b, 2);       // s > 0, "Te" sucede "TE" na ordem
28                                 // lexicográfica
29
30     // Cópia
31     strcpy(a, b);               // a = "TEP"
32     strncpy(a, "SA", 2);        // a = "SAP"
33
34     // Acesso aos elementos individuais
35     a[2] = 'T';                 // a = "SAT"
36     a[0] = b[2];                // a = "PAT"
37
38     // Conversão de string para valores numéricos
39     strcpy(a, "123.45");
40     d = atof(a);                 // d = 123.45
41     i = atoi(a);                // i = 123
42
```


Exemplo de uso dos arquivos string.h e stdlib.h

```

43 strcpy(a, "10000000000000000000000000000000000000000000");
44 ll = strtoll(a, NULL, 2);    // ll = 1099511627776, conversão de base
45                               // binária
46
47 // Concatenação
48 strcpy(b, "xyz");           // b = "xyz"
49 strcpy(a, "abcba");         // a = "abcba"
50
51 strcat(a, b);               // a = "abcbaxyz"
52 strncat(a, a, 3);           // a = "abcbaxyzabc"
53
54 // Busca de caracteres
55 p = strchr(a, 'b');          // p - a = 1, índice da primeira ocorrência
56                               // de 'b'
57 p = strrchr(a, 'b');         // p - a = 9, índice da última ocorrência
58                               // de 'b'
59 p = strstr(a, "cba");        // p - a = 2, índice da primeira ocorrência
60                               // de "cba"
61 p = strstr(a, "dd");         // p = NULL, "dd" não é substring de a
62

```

Exemplo de uso dos arquivos string.h e stdlib.h

```
63     i = strstr(a, "abc");    // i = 5, a[0..4] contém apenas caracteres
64                             // em "abc"
65     i = strcspn(a, "z");    // i = 7, a[0..6] contém caracteres
66                             // diferentes de "z"
67
68     return 0;
69 }
```

Exemplo de uso do arquivo ctype.h

```
1 #include <stdio.h>
2 #include <ctype.h>
3
4 int main()
5 {
6     char a[] = "Test with numbers: 0x1234";
7     int r;
8
9     a[0] = tolower(a[0]);           // a = "test with numbers: 0x1234"
10    a[10] = toupper(a[10]);          // a = "test with Numbers: 0x1234"
11
12    r = isalpha(a[0]);               // r != 0, 't' é alfabético
13    r = isalpha(a[19]);              // r = 0, '\0' não é alfabético
14    r = isalnum(a[19]);              // r != 0, '\0' é alfanumérico
15    r = isblank(a[18]);              // r != 0, ' ' é um espaço em branco
16    r = isdigit(a[1]);               // r = 0, 'e' não é um dígito decimal
17    r = isxdigit(a[1]);              // r != 0, 'e' é um dígito hexadecimal
18
19    return 0;
20 }
```

Strings em C++

Representação de strings em C++

- Embora seja possível utilizar a abordagem e as bibliotecas da linguagem C em C++, existe uma representação em C++ de strings baseada em classes
- O uso de classes para representar strings traz a vantagem de poder manter outras informações sobre a string sempre atualizadas e com acesso em $O(1)$
- Por outro lado, esta representação demanda mais memória do que a representação em C
- Existem técnicas para tentar reduzir a memória utilizada, como a *small string optimization*
- A classe fundamental dentre as várias classes que representam strings em C++ é a `std::string`.
- O método `c_str()` permite obter, a partir de uma string C++, uma representação compatível com a utilizada em C
- Deste modo, é possível utilizar funções escritas para strings em C a partir de instâncias da classe da linguagem C++

Exemplo de uso da classe string em C++

```
1 #include <iostream>
2
3 using namespace std;
4
5 int main()
6 {
7     string a { "Test" }, b { "TEP" };
8
9     // Tamanho da string
10    int s = a.size();      // s = 4
11    s = b.size();          // s = 3
12
13    // Comparação
14    bool r = (a == b);     // r = false, "Test" e "TEP" são distintas
15    r = (a != b);          // r = true, "Test" e "TEP" são distintas
16    r = (a < b);           // r = false, "Test" sucede "TEP" na ordem
17                           // lexicográfica
18    r = (a > b);           // r = true, "Test" sucede "TEP" na ordem
19                           // lexicográfica
20
```

Exemplo de uso da classe string em C++

[illegible]

Exemplo de uso da classe string em C++

```
42
43 // Conversão de valores numéricos para strings
44 a = to_string(999);    // a = "999"
45 a = to_string(9.99);   // a = "9.99"
46
47 // Concatenação
48 b = "xyz";             // b = "xyz"
49 a = "abcba";           // a = "abcba"
50
51 a += b;                // a = "abcbaxyz"
52 a += a.substr(0, 3);   // a = "abcbaxyzabc"
53
54 // Busca de caracteres
55 auto p = a.find('b');   // p = 1, índice da primeira
56                        // ocorrência de 'b'
57 p = a.rfind('b');       // p = 9, índice da última
58                        // ocorrência de 'b'
59 p = a.find("cba");       // p = 2, índice da primeira
60                        // ocorrência de "cba"
61 p = a.find("dd");       // p = string::npos, "dd" não é
62                        // substring de a
```


Exemplo de uso da classe string em C++

```
63
64     p = a.find_first_not_of("abc");    // i = 5, a[0..4] contém apenas
65                                         // caracteres em "abc"
66     p = a.find_first_of("z");          // i = 7, a[0..6] contém
67                                         // caracteres diferentes de "z"
68
69     // Exemplo de uso do método c_str()
70     a = "Test";
71     printf("%s\n", a.c_str());
72
73     return 0;
74 }
```

Strings em Python

Representação de strings em Python

- Embora as strings em Python também sejam implementadas através de classes, elas podem ser vistas informalmente como listas de caracteres
- Em Python, constantes do tipo string podem ser representados usando-se aspas simples ou duplas, ou mesmo triplas (para strings com múltiplas linhas)
- Para maratonas de programação, o módulo string da linguagem Python traz constantes bastantes úteis, como listagens de caracteres comuns:

```
import string
```

```
a = string.lowercase      # a = 'abcdefghijklmnopqrstuvwxyz'
b = string.uppercase      # b = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
c = string.letters        # c = a + b
d = string.digits         # d = '0123456789'
x = string.hexdigits      # x = '0123456789abcdefABCDEF'
```

Representação de strings em Python

- Mesmo que a solução proposta pela equipe seja escrita em outra linguagem, estas constantes podem ser facilmente acessadas via terminal, importando o módulo e usando o comando (ou função, no Python 3) **print**
- Outra particularidade do Python é que, ao contrário das linguagens C e C++, ele suporta índices negativos para strings
- Por exemplo, `s[-1]` se refere ao último caractere, `s[-2]` ao penúltimo, e assim por diante
- Outra notação útil é `s[::-1]`, que indica o reverso da string `s` (isto é, `s` lida do fim para o começo)
- A API para strings em Python contempla ainda muitas outras funções úteis, como `strip()`, `join()` e `split()`

Exemplo de uso de strings em Python

```
1 # -*- coding: utf-8 -*-
2
3 if __name__ == '__main__':
4
5     a = "Test"
6     b = "TEP"
7
8     # Tamanho da string
9     s = len(a)          # s = 4
10    s = len(b)          # s = 3
11
12    # Comparação
13    r = (a == b)         # r = False, "Test" e "TEP" são distintas
14    r = (a != b)         # r = True, "Test" e "TEP" são distintas
15    r = (a < b)          # r = False, "Test" sucede "TEP" na ordem
16                        # lexicográfica
17    r = (a > b)          # r = True, "Test" sucede "TEP" na ordem
18                        # lexicográfica
19
20    s = a[:1] == b[:1]   # s = True, as strings são iguais no
21                        # primeiro caractere
```


Exemplo de uso de strings em Python

```
43
44 # Concatenação
45 b = "xyz"          # b = "xyz"
46 a = "abcba"        # a = "abcba"
47
48 a += b             # a = "abcbaxyz"
49 a += a[:3]         # a = "abcbaxyzabc"
50
51 # Busca de caracteres
52 p = a.find('b')     # p = 1, índice da primeira ocorrência de 'b'
53 p = a.rfind('b')    # p = 9, índice da última ocorrência de 'b'
54 p = a.find("cba")   # p = 2, índice da primeira ocorrência de "cba"
55 p = a.find("dd")    # p = -1, "dd" não é substring de a
56
57 # Exemplo de uso do método strip()
58 a = "  Espaços antes e depois  "
59 b = a.strip()       # b = "Espaços antes e depois"
60
```

Exemplo de uso de strings em Python

```
61 # Exemplo de uso do método join()
62 xs = ["1", "2", "3", "4", "5"]
63 a = ', '.join(xs)    # a = "1, 2, 3, 4, 5"
64
65 t = ['22', '05', '43']
66 b = ':'.join(t)      # b = "22:05:43"
67
68 # Exemplo de uso do método split()
69 a = "Frase com quatro palavras"
70 b = a.split()        # b = ['Frase', 'com', 'quatro', 'palavras']
71
72 a = "abacad"
73 b = a.split('a')     # b = ['', 'b', 'c', 'd']
74
75 # Exemplo de uso de métodos de alteram o case dos caracteres
76 a = "Tep"
77 b = a.lower()        # b = "tep"
78 c = a.upper()        # c = "TEP"
79 d = a.swapcase()     # d = "tEP"
```


Entrada e saída de string em console

I/O de strings em C

- Cada linguagem tem mecanismos apropriados para a leitura e escritas de strings a partir do terminal
- Em C, são utilizadas as funções `printf()` e `scanf()`
- O marcador utilizado para o tipo string é o `%s`
- A função `scanf()` fará a leitura da entrada até encontrar um caractere de espaço (quebra de linha, tabulações, espaço em branco, etc)
- Se a intenção é ler uma linha na íntegra, deve ser utilizada a função `fgets()`
- A função `fgets()` é mais segura que a `scanf()`, pois utiliza o segundo parâmetro como limite máximo de caracteres (mais o zero terminador) a serem lidos e escritos no primeiro parâmetro
- Vale notar que a função `fgets()` insere, no primeiro parâmetro, o caractere de nova linha, se o encontrar (a função também termina se for encontrado o caractere EOF, que indica o fim do arquivo)

Exemplo de I/O de strings em C

```
1 #include <stdio.h>
2 #include <string.h>
3
4 // Assuma que será inserida em uma linha, via console, a mensagem
5 // "Teste de I/O em C"
6 int main()
7 {
8     char s[1024], line[1024];
9
10    scanf("%s", s);
11    printf("s = [%s]\n", s);    // s = [Teste]
12
13    fgets(line, 1024, stdin);
14    line[strlen(line) - 1] = 0;
15
16    printf("line = [%s]\n", line); // line = [Teste de I/O em C\n]
17
18    return 0;
19 }
```

- Em C++, strings podem ser lidas e escritas com os operadores << e >> das classes cin e cout, respectivamente
- A classe cin se comporta de forma semelhante à função scanf(), lendo a entrada até encontrar um caractere que indique um espaço em branco
- Para ler linhas inteiras, de forma semelhante à fgets(), basta usar a função getline()
- Porém, diferentemente da função fgets(), a função getline() despreza o caractere de fim de linha, e não o insere na string apontada pelo segundo parâmetro

Exemplo de I/O de strings em C++

```
1 #include <iostream>
2
3 using namespace std;
4
5 // Assuma que será inserida em uma linha, via console, a mensagem
6 // "Teste de I/O em C++"
7 int main()
8 {
9     string s, line;
10
11     cin >> s;
12     cout << "s = [" << s << "]\n"; // s = [Teste]
13
14     getline(cin, line);
15     cout << "line = [" << line << "]\n"; // line = [Teste de I/O em C++]
16
17     return 0;
18 }
```

I/O de strings em Python

- Em Python 2, strings podem ser lidas e escritas por meio da função `raw_input()` e pelo comando `print`
- A função `raw_input()` se comporta de maneira semelhante à função `getline()` do C++
- O comando `print` insere, automaticamente, uma quebra de linha após a impressão de sua mensagem
- Para suprimir este comportamento, deve-se usar uma vírgula ao final da mensagem, a qual substitui a quebra de linha por um espaço em branco
- Em Python 3, a função `raw_input()` foi renomeada para `input()`, e o comando `print` foi substituído pela função `print()`

Exemplo de I/O de strings em Python

```
1 # -*- coding: utf-8 -*-
2
3 # Assuma que será inserida em uma linha, via console, a mensagem
4 # "Teste de I/O em Python"
5
6 s = raw_input()
7 print 's = {}'.format(s) # s = [Teste de I/O em Python]
8
9 s = s.split()[0]         # s = "Teste"
10 print s
```

1. **AHLGREEN**, John. [Small String Optimization and Move Operators](#), acesso em 16/12/2016.
2. CppReference. [Null-terminated byte strings](#), acesso em 21/12/2016.
3. CppReference. [std::basic_string](#), acesso em 21/12/2016.
DAVID. [A look at std::string implementations in C++](#), acesso em 22/12/2016.
4. **CROCHEMORE**, Maxime; **RYTTER**, Wojciech. *Jewels of Stringology: Text Algorithms*, WSPC, 2002.
5. **HALIM**, Steve; **HALIM**, Felix. *Competitive Programming 3*, Lulu, 2013.
6. Python Documentation. [Common string operations](#), acesso em 26/12/2016.