UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

Strings

Aula 10

Marcos Silvano Almeida
marcossilvano@utfpr.edu.br
Departamento de Computação
UTFPR Campo Mourão

Strings

- Até o momento, utilizamos **textos literais** nas funções printf() e scanf()

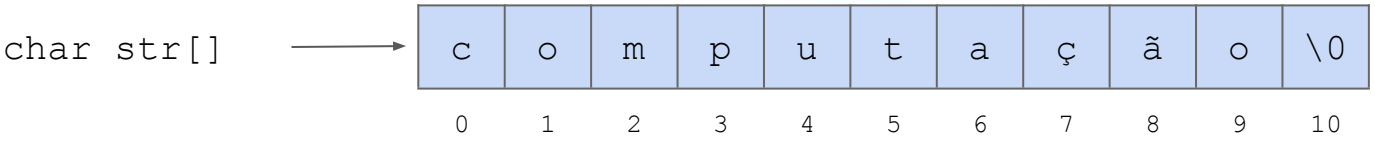
```
scanf("%d %f", &a, &b);
printf("O resultado é %d: ", res);
```

- O tipo texto é chamado de **string**
  - Na linguagem C, uma string é um vetor de caracteres **terminado em nulo:**
    - **'\0'** (char) ou **0** (int)
    -

- Declaração

```
char nome_var_string[tamanho] = "string literal";

char str[] = "computação";  // 10 + \0
```



String e vetores

- Declarando (e inicialização) strings

```
// inicializador de string (adiciona \0 ao final)
char str1[] = "computacao"; // 10 caracteres + \0

char str2[21]; // 20 caracteres + \0

// possível utilizar inicializar de vetor (INCOMUN)
char str3[] = {'c','o','m','p','u','t','a','c','a','o','\0'};
```

- Como uma string é um vetor, devemos tomar os memos cuidados
  - ❌ str[-4] = ‘L’ ⇐ Acessar índices negativos
  - ❌ str[12] = ‘K’; ⇐ Acessar elementos foram do intervalo
  - ❌ str1 = str2; ⇐ Não existe atribuição (cópia) entre vetores
  - ❌ str1 = “testando”; ⇐ Não existe atribuição (cópia) de literais para vetores

Manipulação de Strings

- Da mesma forma que em vetores, podemos criar nossas próprias funções para manipular **strings**
  - Calcular tamanho, copia uma string para outra, passar todas para maiúsculas…
  - Devemos sempre lembrar que: **toda manipulação de string deve obedecer à regra do terminador nulo '\0'**

- Exemplo simples: encontrar o comprimento de uma string

```
char str[] = "string de teste";

int length = 0;
for (int i = 0; str[i] != '\0'; i++) {
    length++;
}
printf("length de str: %d\n", length);
```

A string C é terminada em '\0'

Manipulação de strings

- Passando strings para funções
  - Ao contrário de um vetor habitual, não é necessário passar o tamanho da string para funções, uma vez que sempre termina em '\0'

- Exemplo: encontrar o comprimento de uma string
  - Agora como uma função reutilizável

```
int string_length(char s[]) {
    int length = 0;
    for (int i = 0; s[i] != '\0'; i++) {
        length++;
    }
    return length;
}
```

Usando: `printf("length de str: %d\n", string_length(str));`

Manipulação de strings: outro exemplo

- Suponha que precisamos de uma função que imprima a string com traços

```
void print_dashed_string(char s[]) {
   for (int i = 0; s[i] != '\0'; i++) {
            printf("%c-", s[i]);
        }
   }
   printf("\n");
}
Chamada: char str[] = "string de teste";
         print_dashed_string(str);
Saída: s-t-r-i-n-g- -d-e- -t-e-s-t-e-
```

Pergunta:

> Podemos melhorar a função, impedindo hífens em espaços e ao final?

Biblioteca string.h

Manipulação de Strings: **lib <string.h>**

- Da mesma forma que em vetores, podemos criar nossas próprias funções para manipular **strings**
  - Calcular tamanho, copia uma string para outra, passar todas para maiúsculas…
  - Devemos sempre lembrar que: **toda manipulação de string deve obedecer à regra do terminador nulo '\0'**

- Para simplificar nossa vida, a lib <string.h> provê algumas funções para manipulação básica de strings em C:

`int strlen(char s[])` - encontra o comprimento da string

`strcpy(char dest[], char src[])` - copia uma string em outra

`int strcmp(char s1[], char s2[])` - compara duas strings

`strcat(char dest[], char src[])` - anexa string ao final de outra

Lib <string.h>: exemplos

int strlen(char s[]) - encontra o comprimento da string
strcpy(char dest[], char src[]) - copia uma string em outra

```
char str1[] = "c programming"; // 13  + \0
char str2[40];

strcpy(str2, str1); // equivalente a str2 = str1;
int len = strlen(str2);
printf("str2(%d): %s\n", len, str2);

strcpy(str2, "computer programming"); // 20 + \0
len = strlen(str2);
printf("str2(%d): %s\n", len, str2);
```

Lib <string.h>: exemplos

int strcmp(char s1[], char s2[]) - compara duas strings

```
char str1[] = "advantage";
char str2[] = "advance";
int cmp = strcmp(str1, str2); // cmp < 0, cmp == 0, cmp > 0
if (cmp < 0) { // cmp negativo
    printf("str1 < str2\n"); // str1 vem antes de str2 (alfabeticamente)
}
else {
    if (cmp == 0) { // cmp = 0
        printf("str1 == str2\n"); // str1 é igual a str2
    }
    else { // cmp positivo
        printf("str1 > str2\n"); // str2 vem antes de str1
    }
}
```

Lib <string.h>: mais alguma funções

strcat(char dest[], char src[]) - anexa string ao final de outra

```c
int main() {
   char str1[20] = "c ";
   char str2[] = "programming";

   // concatena (anexa) str2 ao final de str1
   // OBS: str1 deve ter espaço suficiente
   strcat(str1, str2);
   printf("str1 + str2: %s\n", str1);

   return 0;
}
```

Considerações sobre a lib <string.h>

- As funções que vimos da string.h provêm utilitários para manipulação de strings
  - Entretanto, não são soluções para todos os problemas
  - Vamos utilizá-las quando adequado. Em muitos outro casos, vamos implementar nossas próprias funções de manipulação

- Sempre devemos lembrar que:
  - Uma string válida em C é um **vetor de caracteres terminado em nulo '\0'**
  - Só conseguimos acessar a string por meio de seus caracteres
    - Portanto, vamos precisar de (ao menos) um laço para percorrer uma string

```
// percorre a string, acessando cada caractere e encerrando no '\0'
for (int i = 0; s[i] != '\0'; i++) {
    // faz algo com o caractere corrente em s[i]
}
```

Strings e scanf()

String e scanf()

- Toda manipulação de string deve obedecer à regra do terminador nulo ‘\0’
  - Desta forma, a string será válida na linguagem C
  - Todas as funções da linguagem que lidam com strings consideram tal regra

```
char str[10]; // 9 caracteres + \0
scanf("%s", str); // OK para texto de até 9 chars e sem espaços...
printf("%s\n", str);
```

- Problemas com scanf() para leitura de strings
  - Encerra em ENTER ou ESPAÇO
  - Não limita a quantidade de caracteres considerados na entrada
    - Ex: char nome[10] ⇐ não podemos digitar mais que 9 chars

String, char e scanf(): solução

- Podemos usar uma **expressão regular** para informar que a função scan() deve ler somente X caracteres e tratar espaços como texto

```
char s[10];

scanf(" %9[^\n]", s); // lê 9 caracteres, encerra em \n e adiciona \0
```

Não utilizamos **&** para informar o endereço da variável string (variáveis de vetores sempre indicam o endereço)

- Entretanto, ainda temos o problema de caracteres que ficam no buffer de entrada e são consumidos pelo próximo scanf() automaticamente.
  - Para tanto, usamos uma função para consumir os caracteres restantes no buffer após chamada de scanf() de entrada de caractere `" %c"` ou string `" %9[^\n]"`.

```
// utilizar após scanf ou getchar
void clear_buffer() {
    while (getchar() != '\n');
}
```

String, char e scanf(): exemplo

```c
char product[21]; // 20 caracteres + \0
printf("\nProduto [20]:\n> ");
scanf(" %20[^\n]s", product);
clear_buffer();

printf("\n[C]PU  [G]PU  [R]am  [M]otherboard  [S]torage? \n> ");
char type;
scanf(" %c", &type);
clear_buffer();

char description[101]; // 100 caracteres + \0
printf("\nDescrição [100]:\n> ");
scanf(" %100[^\n]s", description);
clear_buffer();

printf("\nRESUMO:\n");
printf("  Produto: %s  (%c)\n  Descricao: %s\n", product, type, description);
```

Utilizamos a função clearBuffer() após cada leitura de string ou caractere.

UTFPR

Referências

- Algoritmos e Programação
  - Marcela Gonçalves dos Santos
  - Disponível pelo Moodle

- Estruturas de Dados, Waldemar Celes e José Lucas Rangel
  - PUC-RIO - Curso de Engenharia
  - Disponível pelo Moodle

- Linguagem C, Silvio do Lago Pereira
  - USP - Instituto de Matemática e Estatística
  - Disponível pelo Moodle

- Curso Interativo da Linguagem C
  - https://www.tutorialspoint.com/cprogramming