



Treinamento para a Olimpíada Brasileira de Informática

Exercícios envolvendo strings

Prof. Ciro Cirne Trindade



Strings

- String é um vetor do tipo **char** terminado pelo caractere nulo ('\0')
- Cada caractere de uma string ocupa 1 byte de memória e o último caractere é sempre '\0' (NULL)
- O caractere NULL ou '\0' tem o valor 0 (zero) decimal na tabela ASCII
 - Note que isto não é o mesmo que o caractere '0' que tem valor 48 decimal

String Constante

- Qualquer coisa entre aspas duplas
- Exemplo:

```
printf("Boa aula!");
```

- “Boa aula!” é uma string constante

B	o	a		a	u	l	a	!	\0
---	---	---	--	---	---	---	---	---	----

Variáveis String

- Declaramos uma string em C como um vetor de caracteres
- Exemplo: **char nome[15];**
 - A variável **nome** tem espaço para **14** caracteres, já que toda *string* termina com `'\0'`
 - Cada caractere de uma string pode ser acessado como um elemento do vetor do tipo `char`

Entrada do Tipo String (1/3)

- `scanf`
 - Permite delimitar o tamanho da entrada
 - Não aceita espaços em branco (pode ser contornado através de um *scanset*)
 - Exemplo: `scanf(" %14[^\n]", nome);`

Tamanho
máximo da
entrada

scanset que indica que
qualquer caractere é
válido na entrada, com
exceção (^) do Enter (\n)

Entrada do Tipo String (2/3)

- `fgets`
 - Permite delimitar o tamanho da entrada, mas pode armazenar um `'\n'` no final da string
 - Aceita espaços em branco
 - Protótipo: `char * fgets(char *, int, FILE *);`
 - Exemplo: `fgets(nome, 15, stdin);`

Tamanho máximo
da entrada

`stdin` (*standard input* – entrada padrão): *stream* que representa o teclado



Entrada do Tipo String (3/3)

- `gets`
 - Não permite delimitar o tamanho da entrada
 - Aceita espaços em branco
 - Protótipo: `char * gets(char *);`
 - Exemplo: `gets(nome);`



Saída do Tipo String (1/2)

- `printf`
 - Exemplo: `printf("Saudacoes, %s\n", nome);`
- `puts`
 - Exibe uma string no vídeo e salta uma linha
 - Aceita apenas um argumento
 - Exemplo: `puts(nome);`

Saída do Tipo String (2/2)

- `sprintf`

- Permite a concatenação de informações de tipos diferentes em uma string

- Protótipo:

```
int sprintf(char *, const char *, ...);
```

- Exemplo:

```
sprintf(msg, "Valor de PI = %lf", M_PI);
```

Irá conter a string:
"Valor de PI = 3.151593"

Inicializando Strings

- Caractere a caractere

```
char nome[] = { 'A', 'n', 'a', '\0' };
```

- Toda de uma vez

```
char nome[] = "Ana";
```

- String constante

```
char * nome;
```

```
nome = "Ana";
```

Ponteiro

Varrendo uma string caractere a caractere

- Para varrer uma string caractere a caractere, fazemos um `for` até que o caractere `'\0'` seja encontrado
- Exemplo:

```
for (i = 0; str[i] != '\0'; i++) {  
    putchar(str[i]);  
}
```



Funções de Manipulação de Strings (1/11)

- A biblioteca do C provê várias funções de manipulação de strings
- Estas funções estão definidas no arquivo **string.h**



Funções de Manipulação de Strings (2/11)

- `strlen()`
 - Retorna o comprimento de uma string (número de caracteres, sem contar o `'\0'`)
 - Protótipo:
`int strlen(const char *);`



Exemplo do uso de `strlen()`

```
#include <stdio.h>
#include <string.h>
int main() {
    char str[41];
    int comp;
    printf("Informe uma string: ");
    scanf("%40[^\n]", str);
    comp = strlen(str);
    printf("A string tem %d caracteres\n",
           comp);
    return 0;
}
```



Funções de Manipulação de Strings (3/11)

- `strcat()`
 - Concatena duas strings
 - Aguarda dois argumentos do tipo string e concatena no final da 1ª o conteúdo da 2ª
 - Protótipo:

```
char * strcat(char *, const char *);
```



Exemplo do uso de `strcat()`

```
#include <stdio.h>
#include <string.h>
int main() {
    char flor[41];
    char sufixo[] = " cheira como sapato velho";
    printf("Qual sua flor favorita? ");
    scanf("%40[^\n]", flor);
    strcat(flor, sufixo);
    puts(flor);
    return 0;
}
```

Pode extrapolar o
tamanho da string `flor`

Funções de Manipulação de Strings (4/11)

- `strncat()`

- A função `strncat()` não verifica se a 2ª string cabe na 1ª
- Através da função `strncat()` é possível informar o número máximo de caracteres da 2ª string que deve ser adicionado à 1ª
- Protótipo:

```
char * strncat(char *, const char *,  
int);
```



Funções de Manipulação de Strings (5/11)

- `strcmp()`
 - Não é possível comparar 2 strings através dos operadores relacionais
 - A função `strcmp()` espera duas strings como argumentos e compara-as lexicograficamente devolvendo:
 - 0: se as duas strings são iguais
 - < 0 : se a 1ª string vem antes da 2ª
 - > 0 : se a 1ª string vem depois da 2ª
 - Protótipo:

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



Exemplo do uso de `strcmp()`

```
#include <stdio.h>
#include <string.h>
int main() {
    char tentativa[21];
    char resposta[] = "Cabral";
    printf("Quem descobriu o Brasil? ");
    scanf(" %20[^\n]", tentativa);
    while (strcmp(tentativa, resposta) != 0) {
        puts("Nao, esta' errado. Tente novamente");
        scanf(" %20[^\n]", tentativa);
    }
    puts("Esta' certo!");
    return 0;
}
```



Funções de Manipulação de Strings (6/11)

- `strncmp ()`
 - Esta função compara apenas os n primeiros caracteres da duas strings passadas como argumentos
 - Protótipo:

```
int strncmp(const char *, const char *,  
int);
```



Exemplo do uso de `strncmp()`

```
#include <stdio.h>
#include <string.h>
int main() {
    char tentativa[21];
    printf("Digite uma palavra começando com para: ");
    scanf(" %20[^\n]", tentativa);
    while (strncmp(tentativa, "para", 4) != 0) {
        puts("Nao, esta' errado. Tente novamente");
        scanf(" %20[^\n]", tentativa);
    }
    printf("Palavra aceita: %s\n", tentativa);
    return 0;
}
```



Funções de Manipulação de Strings (7/11)

- `strcasecmp()` e `strncasecmp()`
 - As funções `strcasecmp()` e `strncasecmp()` são equivalentes às funções `strcmp()` e `strncmp()`, respectivamente, porém consideram letras maiúsculas e minúsculas iguais (*case insensitive*)
 - Não são padrão ISO/ANSI



Funções de Manipulação de Strings (8/11)

- `strcpy()`
 - Não é possível atribuir uma string a outra através do operador de atribuição (`=`)
 - Para este fim utilize a função `strcpy()`
 - Protótipo:

```
char * strcpy(char * s1,  
               const char * s2);
```
 - Equivalente a fazer `s1 = s2`



Exemplo do uso de `strcpy()`

```
#include <stdio.h>
#include <string.h>
int main() {
    char palavra[21], copia[21];
    printf("Digite uma palavra: ");
    scanf(" %20[^\n]", palavra);
    strcpy(copia, palavra);
    printf("Copia da palavra: %s\n", copia);
    return 0;
}
```




Funções de Manipulação de Strings (9/11)

■ `strncpy ()`

- A função `strcpy ()` não verifica se a 2ª string cabe na 1ª
- Para copiar apenas os n primeiros caracteres da 2ª string para a 1ª, utilize a função `strncpy ()`

■ Protótipo:

```
char * strncpy(char * s1,  
               const char * s2, int n);
```

- Não coloca o `'\0'` no final de `s1`



Exemplo do uso de `strncpy()`

```
#include <stdio.h>
#include <string.h>
int main() {
    char palavra[21], copia[21];
    int n;
    printf("Digite uma palavra: ");
    scanf(" %20[^\n]", palavra);
    printf("Quantas letras quer copiar? ");
    scanf("%d", &n);
    strncpy(copia, palavra, n);
    copia[n] = '\0';
    printf("Copia da palavra: %s\n", copia);
    return 0;
}
```



Funções de Manipulação de Strings (10/11)

- `strstr()`

- Procura uma string dentro de outra

- Protótipo:

```
char * strstr(const char * s1,  
              const char * s2);
```

- Devolve um ponteiro para o início de `s2` em `s1`
- Devolve `NULL` se `s2` não estiver contida em `s1`



Exemplo do uso de `strstr()`

```
#include <stdio.h>
#include <string.h>
int main() {
    char str1[21], str2[21];
    printf("Digite duas palavras: ");
    scanf(" %20[^\n] %20[^\n]", str1, str2);
    if (strstr(str1, str2) != NULL) {
        printf("%s ocorre em %s\n", str2, str1);
    }
    else {
        printf("%s NAO ocorre em %s\n", str2, str1);
    }
    return 0;
}
```



Funções de Manipulação de Strings (11/11)

- `strtok ()`

- Quebra uma string em uma série de tokens usando um delimitador
- Protótipo:

```
char * strtok(char * str, const char *  
delim);
```

- `str`: o conteúdo dessa string é modificado e quebrado em tokens menores (strings)
- `delim`: o delimitador dos tokens em `str`
- Devolve um ponteiro para o último token encontrado em `str`
- Devolve `NULL` se não houver mais nenhum token



Exemplo do uso de `strtok()`

```
#include <stdio.h>
#include <string.h>

int main()
{
    char str[80] = "olimpiada;brasileira;informatica";
    char s[] = ";";
    char * token;

    /* pega o 1º token */
    token = strtok(str, s);

    /* pega os outros tokens */
    while (token != NULL) {
        printf("%s\n", token );
        token = strtok(NULL, s);
    }

    return 0;
}
```

Maiúsculo e minúsculo

- Não existe uma função padrão ISO/ANSI que converta uma string para maiúsculo ou minúsculo, mas existem funções que devolvem o caractere maiúsculo e minúsculos correspondentes a um caractere
- `char toupper(char c);`
 - Devolve o caractere maiúsculo correspondente a `c`
- `char tolower(char c);`
 - Devolve o caractere minúsculo correspondente a `c`
- Definidas em `ctype.h`



Exercícios

- Telefone: OBI2008 – 1ª Fase –
Modalidade Programação, Nível 2
 - http://olimpiada.ic.unicamp.br/pratique/programacao/nivel2/2008f1p2_telefone
- Auto Estrada: OBI2008 – 2ª Fase –
Modalidade Programação, Nível 2
 - http://olimpiada.ic.unicamp.br/pratique/programacao/nivel2/2008f2p2_auto



Vetor de Strings

- Matriz de caracteres

- Por exemplo:

```
char str_array[30][80];
```

- Declara uma matriz de 30 *strings*, cada qual com comprimento máximo de 79 caracteres
- Para acessar uma *string* individual na matriz, basta especificar o 1º índice (linha)

```
scanf(" %79[^\n]", str_array[2]);
```

- Faz referência a 3ª *string* em **str_array**



Inicializando um Vetor de Strings

- Forma tradicional: matriz bidimensional de caracteres

```
char naipes[4][8] = { "Copas", "Ouros",  
                      "Paus", "Espadas" };
```

- Ocupa 32 bytes de memória
- Vetor de ponteiros para caractere

```
char * naipes[4] = { "Copas", "Ouros",  
                    "Paus", "Espadas" };
```

- Ocupa 25 bytes de memória



Exercícios

- Costa: OBI2012 – 2ª Fase – Modalidade Programação – Nível 1
 - http://olimpiada.ic.unicamp.br/pratique/programacao/nivel1/2012f2p1_costa
- Batalha Naval: OBI2010 – 1ª Fase – Modalidade Programação – Nível 2
 - http://olimpiada.ic.unicamp.br/pratique/programacao/nivel2/2010f1p2_batalha



Referências

- UNICAMP. *Olimpíada Brasileira de Informática*. Disponível em:
<<http://http://olimpiada.ic.unicamp.br>>.