

Aula 14

MC 102 - Algoritmos e Programação de Computadores

Strings: cadeia de caracteres.

Strings (cadeias de caracteres)

Uma cadeia de caracteres (**char**), ou **string**, é uma seqüência de letras e símbolos (dígitos, ?, !, & *, etc).

Já utilizamos **strings** para definir textos para impressão em nossos programas.

```
printf("Oi mundo.\n");
```

Mas o que usamos para representar uma **string**?

- Strings se aproximam de um vetor de caracteres.

```
#define TAMANHO 15;
```

```
char hello_world[TAMANHO+1] = "Oi mundo!";
```

Strings (cadeias de caracteres)

Por que TAMANHO+1 ?

- Em C, uma cadeia de caracteres é terminada com o marcador especial: **'\0'**.
- Este marcador indica o final da string
- Portanto, utilizar uma posição além do tamanho máximo desejado para que possa ser colocado o marcador **'\0'** no final da maior cadeia armazenável na variável.

Usando strings

Sintaxe:

```
char identificador[ numero_de_caracteres +1 ]
```

Inicialização:

Como nos vetores numéricos, pode ser feita na sua definição

Strings não inicializadas

```

char linha[181];
char nome[31];
char texto[] = "Nome: ";
char *texto2 = "RA: ";

```

Strings inicializadas

Usar ***texto2** ou **texto2[]** significa a mesma coisa na linguagem C. Mais adiante, o significado do ***** será apresentado neste curso.

Usando strings

Sintaxe:

char [identificador] [numero_de_caracteres + 1]

Inicialização:

Como nos vetores numéricos, pode ser feita na sua definição

Strings não inicializadas

```
char linha[48];
char nome[31];
char texto[] = "Nome: ";
char *texto2 = "RA: "
```

Apesar de não ter explicitamente definido o tamanho máximo para a string *texto*, ela só poderá armazenar até 6 caracteres, pois este é o tamanho da constante utilizada em sua inicialização.

Usando strings

Para imprimir uma string, basta utilizar o comando **printf** com a formatação **%s**:

```
char texto[] = "Nome: ";
char nome[31] = "Fulano de Tal";

printf("%s\n", texto);
printf("%s\n", nome);
```

O que acontece se executarmos a seqüência abaixo?

```
int i;
char nome[31] = "Fulano de Tal";

for(i=0; i<31; i++)
    printf("%c", nome[i]);
```

Usando strings

Por ser definida como uma cadeia de caracteres, cada posição pode ser acessada individualmente como um caracter.

Porém, as strings sempre são armazenadas com o caracter **'\0'** em seu final, indicando o seu fim.

```
char nome[31] = "Fulano de Tal";

for(i=0; i<31; i++)
    printf("%c", nome[i]);
```

Logo, a string **nome** só pode armazenar **30** caracteres, pois o último será utilizado para o caracter de fim de string **'\0'**. Além disso, temos de testar o fim da string no for. Como?

Usando strings

Por ser definida como uma cadeia de caracteres, cada posição pode ser acessada individualmente como um caracter.

Porém, as strings sempre são armazenadas com o caracter **'\0'** em seu final, indicando o seu fim.

```
char nome[31] = "Fulano de Tal";

for(i=0; i<31 && nome[i] != '\0'; i++)
    printf("%c", nome[i]);
```

Lendo uma string

Para ler uma string do teclado, podemos fazer de modo semelhante à `printf` e utilizar o formato `%s` para a função `scanf`. Porém, `scanf` oferece uma restrição: **a leitura será realizada até que o usuário digite espaço em branco ou "enter"**.

```
char nome[31];
printf("Digite seu primeiro nome: ");
scanf("%s", nome);
printf("Seja bem-vindo, %s\n", nome);
```

Note que aqui não utiliza-se o `&` para a leitura de cadeia de caracteres. Por quê?

Lendo uma string

Para ler uma string do teclado, podemos fazer de modo semelhante à `printf` e utilizar o formato `%s` para a função `scanf`. Porém, `scanf` oferece uma restrição: **a leitura será realizada até que o usuário digite espaço em branco ou "enter"**.

```
char nome[31];
printf("Digite seu primeiro nome: ");
scanf("%s", nome);
printf("Seja bem-vindo, %s\n", nome);
```

Note que aqui não utiliza-se o `&` para a leitura de cadeia de caracteres. A variável `nome` do tipo `string` já armazena o endereço da cadeia. Lembram disso?

Lendo uma string

Uma forma de contornar a restrição do `scanf` é utilizar outra função:

gets: lê uma string até que o usuário digite "enter".

```
char nome[256];
printf("Digite seu nome completo: ");
gets(nome);
printf("Seja bem-vindo, %s\n", nome);
```

Atenção! Como não é possível especificar o limite para a leitura da string, deve-se tomar cuidado em declarar seu tamanho de forma a caber toda a entrada digitada.

Atribuições em strings

Para alterar um caracter em uma string se faz da mesma forma que um vetor:

```
texto[4] = 'c';
```

Mas como alterar todo o conteúdo da string? Isso é possível sem usar um comando de repetição?

Atribuições em strings

Para alterar um caracter em uma string se faz da mesma forma que um vetor:

```
texto[4] = 'c';
```

Mas como alterar todo o conteúdo da string? Isso é possível sem usar um comando de repetição?

Sim. Para isso deve-se usar a biblioteca **string.h** para se ter acesso às funções que manipulam strings.

Atribuições em strings

Função strcpy:

```
strcpy(char *destino, char *origem);
```

Esta função **copia** todo o **conteúdo** da string **origem** para a string **destino** . Todo o conteúdo de destino é perdido e só o conteúdo de origem permanece.

Exemplos:

```
char str1[] = "Aluno:";
char str2[] = "RA:";
char texto[21];

strcpy(texto, "Fulano de Tal");
printf("%s %s\n", str1, texto);
strcpy(texto, "070000");
printf("%s %s\n", str2, texto);
```

Tamanho de strings

Função strlen:

```
int strlen(char *str1);
```

Retorna a **quantidade** de **caracteres da string** até o caracter '\0'. Não confundir com o tamanho máximo da string

Exemplos:

```
char str1[] = "Tamanho da string";
char str2[21];
int n;

scanf("%s", str2);
n = strlen(str1);
printf("'s' tem tamanho %d\n", str1, n);
n = strlen(str2);
printf("'s' tem tamanho %d\n", str2, n);

//...
```

Concatenando strings

Função strcat:

```
int strcat(char *destino, char *origem);
```

Contatena (insere) a string **origem** no **final** da string **destino** .

Exemplos:

```
char str1[61];
char str2[21];

scanf("%s", str2);
strcpy(str1, "A string digitada e ");
strcat(str1, str2);
strcat(str1, "\n");
printf("%s", str1);
...
```

Comparação de strings - 1

Função strcmp:

```
int strcmp(char *str1, char *str2);
```

compara 2 strings e retorna:

0 (str1 == str2), <0 (str1 < str2), e >0 (str1 > str2)

Exemplos:

```
char str1[21], str2[21];
int res;
scanf("%s %s", str1, str2);
res = strcmp(str1, str2);
if (res > 0)
    printf("%s' e maior que '%s'\n", str1, str2);
else if (res < 0)
    printf("%s' e maior que '%s'\n", str2, str1);
else
    printf("%s' e '%s' são iguais\n", str1, str2);
```

Comparação de strings - 2

Função strcasecmp:

```
int strcasecmp(char *str1, char *str2);
```

compara 2 strings ignorando diferenças entre maiúsculas e minúsculas.

Retorna 0 (str1 == str2), <0 (str1 < str2), e >0 (str1 > str2)

Exemplos:

```
char str1[21], str2[21];
int res;
scanf("%s %s", str1, str2);
res = strcasecmp(str1, str2);
if (res > 0)
    printf("%s' e maior que '%s'\n", str1, str2);
else if (res < 0)
    printf("%s' e maior que '%s'\n", str2, str1);
else
    printf("%s' e '%s' são iguais\n", str1, str2);
```

Convertendo strings

Funções:

```
int atoi(char *str);
```

Converte o conteúdo da string **str** para um valor **inteiro**.

```
double atof(char *str);
```

Converte o conteúdo da string **str** para um valor **double**.

Exemplos:

```
char str[] = "1976";
int i;
double n;
i = atoi(str);
n = atof(str);
printf("%d <--> %f\n", i, n);
...
```

Exercício

Escreva um programa que dadas duas strings, execute as as funções strcpy, strcat e strlen. Atenção: não use a biblioteca string.h

Exercício

```
01001101 01000011 00110001 00110000 00110011 00100000 00101101 00100000 01000001 01101100 01100111 01101111
01110010 01101001 01110100 01101101 01101111 01110011 00100000 01100101 00100000 01010000 01110010 01101111
01100111 01110010 01100001 01101101 01100001 11100011 11100011 01101111 00100000 01100100 01100101 00100000
01000011 01101111 01101101 01110000 01110101 01101111 01110010 01100101 01110011 01100111
00001101 00001010 01010100 01110101 01110011 00100000 01001101 00101100 00100000
01001110 00100000 01100101 00100000 00110101 00100000 00101101 00100000 01110010 01101001 01101101
01100101 01101001 01110010 01101111 00100000 01010011 01100101 01101101 01100101 01110011 01110100 01110010
011#include <stdio.h> 01100101 00100000 00110010 00110000 00110111 00001101 00001010 01010000
01110010 01101111 01100110 01100101 01110011 01110011 01110010 00110000 01001110 01101001
011int main(int argc, char **argv) { 01000000 01010011 01101001 01101101 11110101 01100101 01110011
00001 char str1[11], str2[11]; 01100101 01101110 01110100 01100001 00111010 00001101 00001010 01000110
01110 char strfin1[23], strfin2[11]; 01100101 01101110 01110100 01101111 01110011 00100000 01100100
011100 int tam; 01100001 01101100 01100111 01101111 01110010 01101001 01110100 01101101 01110011 01110011
00100000 01100101 00100000 01110011 01110101 01100001 00100000 01110010 01100101 01110000 01110010 01100101
01110 printf("Informe a primeira string:\n"); 000011 01101111 00100000 01100101 01101101 00100000
01101 gets(str1); 011110 01100111 01110101 01110001 01100111 01100101 01110110 01110011 00100000 01100100
01100 printf("Informe a segunda string:\n"); 01100000 01101110 11101101 01110110 01110010 01101100
00101 gets(str2); 000101 01110011 01110100 01110101 01100100 01101111 00100000 01110000 01101111 01110010
01101101 01100101 01101110 01101111 01110010 01101001 01110110 01100001 01100100 01101111 00100000 01100100
01100 for (i=0; i<11 && str1[i]!='\0'; i++) 01111 01110101 00100000 01101101 01100001 01101001
0111001 strfin2[i] = str1[i]; 01101110 01100111 01100101 01100001 01100111 01100101 01110011 01100110 01110011
00101 printf("String copiada para strfin2: %s", 110 0110110 01101111 01101100 01110110 01101001
01101101 strfin2); 10 01110100 01101111 00100000 01110011 01101001 01110011 01110100 01110010 01101101
11100001 01110100 01100011 01101111 00100000 01100101 00100000 01101001 01101101 01110000 01101100
01100 for (i=0; i<11 && str1[i]!='\0'; i++) 00111 11100011 01101111 00100000 01100100 01100101
00100000 strfin1[i] = str1[i]; 01100111 01110010 01100001 01101101 01100001 01110011 00101110 00100000
01001101 01101111 01101000 01101101 01100001 01110010 01101001 01100100 01100001 01100100 01100101
00101100 for (j=0; j<11 && str2[j]!='\0'; j++) 11100011 11100011 01101111 00101100
00100000 strfin1[i++] = str2[j]; 110100 01100101 01110011 00100000 01100101 00100000 01100100 01101111
01100011 01110101 01101101 01100101 01101110 01110100 01100001 11100011 11100011 01101111 00101100
20. Sem 2007. Algoritmos e Programação de Computadores - Turmas I J K L 21
```

Exercício

```
01001101 01000011 00110001 00110000 00110011 00100000 00101101 00100000 01000001 01101100 01100111 01101111
01110010 01101001 01110100 01101101 01101111 01110011 00100000 01100101 00100000 01010000 01110010 01101111
01100111 01110010 01100001 01101101 01100001 11100011 11100011 01101111 00100000 01100100 01100101 00100000
01000011 01101111 01101101 01110000 01110101 01110000 01110101 01100101 01110011 01100111
00001101 00001010 01010100 01110101 01110011 00100000 01001101 00101100 00100000
01001110 00100000 01100101 00100000 00110101 00100000 00101101 00100000 01010000 01110010 01101001 01101101
01100101 01101001 01110010 01101111 00100000 01010011 01100101 01100101 01110011 01110100 01110010 01110010
011#include <stdio.h> 01100101 00100000 00110010 00110000 00110111 00001101 00001010 01010000
01100101 01101111 01100110 01100101 01110011 01110011 01110010 00110000 01001110 01101001
011int main(int argc, char **argv) { 01000000 01010011 01101001 01101101 11110101 01100101 01110011
00001 char str1[11], str2[11]; 01100101 01101110 01110100 01100001 00111010 00001101 00001010 01000110
01110 char strfin[23]; 01100101 01101110 01110100 01110111 01110011 00100000 01100100
011100 int tam, i, j; 01100001 01101101 01100101 01101101 01110100 01110110 01110111 01110110 01110011 01110011
00100000 printf("Informe a primeira string:\n"); 010001 01101111 00100000 01100101 01101101 00100000
01110 gets(str1); 011110 01110100 01100001 01100101 01101101 00100000
01101 printf("Informe a segunda string:\n"); 01100101 01101110 01110011 00100000 01100100
01100 gets(str2); 000001 01100110 01110100 01110101 01100100 01101111 00100000 01110000 01101111 01110010
00101110 for (i=0; i<11 && str1[i]!='\0'; i++) 01111 01110101 00100000 01101101 01100001 01101001
0111001 strfin[i] = str1[i]; 01101110 01100111 01100101 01100001 01100111 01100101 01110011 01100110 01110011
01110 printf("String copiada para strfin: %s", 110 0110110 01101111 01101100 01110110 01101001
01101110 strfin); 10 01110100 01101111 00100000 01110011 01101001 01110011 01110100 01110010 01101101
11100001 01110100 01100011 01101111 00100000 01100101 00100000 01101001 01101101 01110000 01101100
01100 for (i=0; i<11 && str1[i]!='\0'; i++) 00111 11100011 01101111 00100000 01100100 01100101
00100000 strfin1[i] = str1[i]; 01100111 01110010 01100001 01101101 01100001 01110011 00101110 00100000
01001101 01101111 01101000 01101101 01100001 01110010 01101001 01100100 01100001 01100100 01100101
00101100 for (j=0; j<11 && str2[j]!='\0'; j++) 11100011 11100011 01101111 00101100
00100000 strfin1[i++] = str2[j]; 110100 01100101 01110011 00100000 01100101 00100000 01100100 01101111
01100011 01110101 01101101 01100101 01101110 01110100 01100001 11100011 11100011 01101111 00101100
20. Sem 2007. Algoritmos e Programação de Computadores - Turmas I J K L 22
```