LP1

Prof. Luciano Bernardes de Paula

- Caracteres
- Vetores
- Vetores de caracteres
- **Strings**

Strings

São vetores de caracteres (tipo **char**) que terminam **necessariamente** com um caracter nulo (representado pelo `‘\0’`).

Funções que recebem **strings** utilizam “ ”.

```
printf(“Olá!”);
```

Na memória são armazenados os caracteres seguido de um valor nulo (\0).

“Olá!\0”

0	1	2	3	4
O	l	á	!	\0

Strings como variáveis

É possível ler strings a partir do teclado usando scanf usando o formato %s.

```
...
char str[10];

scanf(“%s”, str);
```

Dessa forma, o scanf lerá todos caracteres entrados até que reconheça um <enter>. Atribuirá todos os caracteres recebidos até o primeiro espaço em branco ou <enter>.

```
scanf(“%s”, str1);
```

Não é preciso colocar o & se for utilizado o nome do vetor sem o índice, pois isso já indica o endereço da primeira posição do vetor.

O scanf não recebe o ‘\n’ (enter) e coloca ‘\0’ no final.

ATENÇÃO!!

Dessa forma, o scanf não trata o tamanho a ser colocado dentro da string.

É altamente recomendado **NÃO** usar o scanf para receber uma string.

Para imprimir na tela uma string usando o printf, faça:

```
char str1[15];

scanf(“%s”, str1);

printf(“%s”, str1);
```

gets()

Essa função armazena todos os caracteres até que o `enter` seja pressionado.

gets(string);

Coloca um ‘\0’ no final da string.

ATENÇÃO!!

O gets não trata o tamanho a ser colocado dentro da string.

É altamente recomendado **NÃO** usar o gets.

Sua versão **mais segura** é a fgets():

```
fgets(str, quantidade, stdin);
```

Essa função recebe no máximo o número de caracteres definido por **quantidade,** vindos do teclado (**stdin**) e sempre insere o ‘\0’ na última posição.

O fgets armazena também um “\n” quando o usuário aperta o Enter e estiver dentro do limite de quantidade, já o gets não.

fgets sempre coloca um ‘\0’ no final da string.

Exemplo:

char str[10];

fgets(str, 10, stdin);

Neste exemplo, serão inseridos no máximo 9 caracteres em str e o último será o ‘\0’.

Se forem inseridos 5 caracteres, por exemplo, o 5º elemento será o ‘\n’ e o 6º será o ‘\0’.

Função útil → sizeof()

A função `sizeof()` recebe uma variável e retorna o tamanho dessa variável em quantidade de bytes.

Como cada variável do tipo **char** ocupa um byte, ao se utilizar um vetor de caracteres, o sizeof retorna exatamente o tamanho desse vetor.

Exemplo:

```
char vetor[10];
int i;
i = sizeof(vetor); // i terá o valor 10
```

Exemplo:

char str[10];

```
fgets(str, sizeof(str), stdin);
```

Outra forma de controlar a entrada de caracteres em uma *string* é obter caracter por caracter e ir preenchendo o vetor.

Exemplo:

Faça um programa que receba caracter por caracter e armazene em uma string e depois apresente na tela.

Funções de manipulação de strings

Para utilizar essas funções é preciso incluir a biblioteca **`string.h`**.

`strlen(string)` – retorna a quantidade de caracteres em uma string.

Exemplo:

```
char string[15] = “Teste!!”;
int i;

i = strlen(string);
```

`strcat(str1, str2)` – concatena a segunda string na primeira.

Cuidado! strcat não verifica se str1 possui tamanho suficiente para acomodar str2.

Exemplo:

```
char str1[10] = “abc”;
char str2[10] = “def”;

strcat(str1, str2);
```

Resultado em str1 = “abcdef”

Uma opção segura:

```
strncat(str1, str2, quantidade);
```

Essa função é mais segura, já que é possível indicar quanto da str2 será alocado em str1.

Lembre-se: str1 pode já possuir caracteres que devem ser contabilizados. Para isso, podemos utilizar a função **strlen()** e o **sizeof()**.

```
strncat(str1, str2, sizeof(str1) – strlen(str1) - 1);
```

Sizeof: quantidade de posições no vetor str1 em bytes;

Strlen: quantidade de caracteres já colocados dentro de str1 até o ‘\0’

-1: para contabilizar um espaço para o ‘\0’ que é colocado pela função

```
strcmp(str1, str2);
```

Retorna um valor:

-1 se a string1 é **menor** que a string 2;

0 as strings são **idênticas**;

1 se a string1 é **maior** que a string2;

Maior e menor aqui significa a ORDEM ALFABÉTICA das strings.

Exemplo:

```
char str1[5];
char str2[5];

fgets (str1, sizeof(str1), stdin);
fgets (str1, sizeof(str1), stdin);

while(strcmp(str1, str2) != 0){
  fgets (str1, sizeof(str1), stdin);
  fgets (str1, sizeof(str1), stdin);
}
```

Dica!

Como na linguagem C não existe variável booleana (que armazenam “verdadeiro” ou “falso”), essas são representadas com valores numéricos.

“0” significa **falso** e “diferente de 0” significa **verdadeiro** (geralmente representado pelo “1”).

É possível, portanto, utilizar funções como abaixo.

```
while (strcmp(str1, str2)){
   ...
}
```

Se as duas strings forem iguais, strcmp retorna 0, fazendo com que o while entenda que o teste foi falso.

Seria equivalente a:

```
while (strcmp(str1, str2) != 0){
   ...
}
```

O mesmo vale para **if** e **do-while**.

`strcpy(str1, str2);`

Copia o conteúdo de str2 em str1.

Cuidado! Não há checagem de tamanho.

`strncpy(str1, str2, quantidade);`

Quantidade indica quantos caracteres de str2 são enviados para str1.

Quantidade pode ser calculada com o **sizeof(str1)**, porém, lembre-se de reservar um espaço para o ‘\0’.

Atenção!

Se a quantidade de caracteres da segunda string não chegar até o ‘\0’ dela, esse **não** é colocado pela função strncpy.

Combinando sizeof com as outras funções

```
fgets(str, sizeof(str), stdin);
```
// fgets **já considera** o sizeof(str) – 1 e coloca um \0 no final da string

```
strncat(str1, str2, sizeof(str1) – strlen(str1) - 1);
```
// o -1 é para guardar uma posição para o \0

```
strncpy(str1, str2, sizeof(str1) - 1);
```
// o -1 é para guardar uma posição para o \0 e deve ser checado

Ao usar uma função que manipule strings, sempre cheque se a mesma coloca ou não o ‘\0’ no final.

É preciso checar também, em funções que recebe strings pelo teclado, se o \n é registrado ou não.

A função **fflush(stdin)** limpa o buffer de entrada do teclado.

Abaixo, se número de caracteres inseridos pelo teclado na primeira chamada do fgets exceder o tamanho de str1, o resto será usado como entrada de str2

```
fgets(str1, sizeof(str1), stdin);
fgets(str2, sizeof(str2), stdin);
```

Para evitar isso, usa-se a função **fflush(stdin)**

```
fgets(str1, sizeof(str1), stdin);
fflush(stdin);
fgets(str2, sizeof(str2), stdin);
```

Exemplo

- Faça um programa que receba duas strings e concatene-as em uma única, respeitando a ordem alfabética.

- Exemplo: se as strings recebidas forem “bbbb” e “aaaa”, a string concatenada será “aaaabbbb”.

Exemplo

- Faça um programa que receba 2 palavras em dois vetores diferentes (vet1 e vet2). O programa deve organizar as strings de forma que a palavra contida em vet1 seja menor que a palavra em vet2, seguindo a ordem alfabética.

Exercícios...