Leitura e escrita de strings em C

1 Strings

Strings é o conceito abstrato de uma sequência de caracteres. Particularmente em C, temos duas representações para as strings: vetor de caracteres ou literais (uma sequência de caracteres expressa entre aspas duplas). O caractere '\0' indica o final da string.

Por exemplo, no seguinte código: "Programando em C"é uma string literal, s é uma string como vetor de caracteres e "String: %s"é outra string literal.

```
char s[] = "Programando em C";
printf("String: %s", palavra);
```

2 Escrita e leitura

Os vetores de caracteres podem ser lidos ou escritos com uma única instrução scanf ou printf. O que é uma situação diferente de vetores de inteiros, por exemplo, para os quais você deve utilizar uma estrutura de repetição para ler cada um dos inteiros que compõe o vetor.

3 Escrita na tela

O especificador de formato para strings é %s. Vimos que o especificador %d exige como argumento um **valor inteiro**, %f exige um **valor real**. Para o %s, o printf exigirá como argumento **o endereço inicial** de onde a sequência de caracteres começa. Como obter o endereço inicial do vetor? Há duas opções: (1) o próprio nome do vetor (2) usando o operador & na primeira posição do vetor:

```
    char palavra[100];
    printf("%s", palavra);
```

```
    char palavra[100];
    printf("%s", &palavra[0]);
```

Com isso é possível inclusive escrever uma string a partir de determinado caractere:

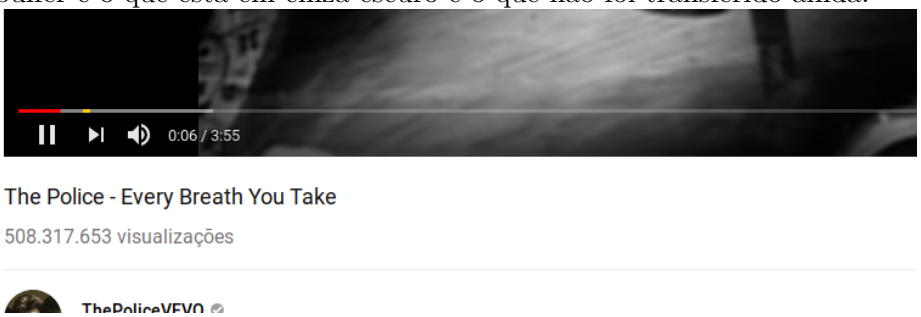
```
    char palavra[100] = "Teste";
    printf("%s", &palavra[2]); //escreve ste na tela
```

4 Leitura

O primeiro cuidado que você deve ter é lembrar que é necessário um espaço no vetor para o '\0'. Então se você declarar um vetor de caracteres com tamanho 100, espere armazenar no máximo 99 caracteres, fora o '\0'.

4.1 Buffer

Outro conceito importante na leitura é o de buffer. Neste contexto, buffer significa um espaço temporário na memória RAM onde os dados são escritos ou de onde os dados são lidos. Para entender melhor o que significa buffer, imagine que você clicou para ver um vídeo do youtube. Nesse momento o vídeo ainda não foi inteiramente transmitido para sua máquina, mas vai sendo transferido e armazenado em um buffer. Quando houver uma quantidade razoável de informações do vídeo no buffer, ele começa a ser reproduzido. Veja na imagem a seguir: na barra de tempo o que está em vermelho é o que já foi reproduzido do vídeo, o que está em cinza claro é o que está no buffer e o que está em cinza escuro é o que não foi transferido ainda.



Na leitura e escrita de dados ocorre o mesmo, as informações vão sendo armazenadas ou lidas a partir de buffer.

4.2 scanf

O scanf faz a leitura de uma string usando o especificador de formato %s. A leitura é realizada até um caractere delimitador (na prática um espaço ou a quebra de linha). Então seja o seguinte código:

```
char palavra[8];
scanf("%s", palavra);
```

Se o usuário digita "Uma palavra", o scanf resgatará do buffer somente "Uma", colocando o '\0' ao final no vetor. O restante do vetor (em vermelho) fica com o que já havia lá antes e o buffer continua com os caracteres não capturados pelo scanf.

i	0	1	2	3	4	5	6	7
palavra[i]	U	m	a	\0				

i	0	1	2	3	4	5	6	7	8
buffer[i]		p	a	l	a	v	r	a	\n

Se o usuário digita "Paralelepipedo"que exige 15 espaços (incluindo \0), então ocorre **buffer overflow**. O scanf não sabe qual é o limite do vetor e sobrescreve as posições seguintes na memória, podendo gerar **falha de segmentação**. A forma de evitar buffer overflow é utilizando a função fgets, descrita mais adiante neste texto.

Uma forma de fazer a leitura de strings com espaços usando o scanf é:

```
char palavra[8];
scanf("%[^\n]", palavra);
```

Esse especificador de formato significa: qualquer sequência desde que não seja um \n. Essa forma de especificador de formato é similar a expressões regulares e é possível fazer várias coisas interessantes com ele, mas para o propósito de ler strings com espaços prefira o uso do fgets (essa última forma do scanf fica mais como curiosidade).

4.3 gets

O gets faz a leitura de uma string considerando os espaços. O único problema dessa função é que, assim como o scanf, não faz a verificação do tamanho do vetor, podendo gerar novamente **falha de segmentação**. Para usar, basta passar como argumento o endereço onde será armazenada a string:

```
char palavra[8];
gets(palavra);
```

Ao compilar com gets você deve receber o seguinte warning do compilador: the 'gets' function is dangerous and should not be used. Esse alerta é justamente pela questão do buffer overflow.

4.4 fgets

O fgets faz a leitura de uma string considerando os espaços e a possibilidade de buffer overflow. Para usar passe como primeiro argumento o endereço onde será armazenada a string. O segundo argumento significa 1 + o número máximo de caracteres a serem lidos do buffer. E o terceiro argumento é a stream (mais adiante na disciplina veremos streams com mais detalhes, mas por enquanto basta colocar stdin).

```
char palavra[8];
fgets(palavra, 8, stdin);
```

4.5 Leitura e escrita de strings na disciplina

Para simplificar tantos detalhes, **na resolução da lista e da prova não haverá perigo de buffer overflow e os limites serão especificados na questão**. Então use uma das duas formas de leitura a seguir:

- se for ler uma palavra(x), isto é, uma sequência de caracteres **sem** espaços use scanf (lembrando que, no mínimo, o \n é deixado no buffer):

  ```
  char palavra[50];
  scanf("%s", palavra);
  ```

- se for ler uma string(x), isto é, uma sequência de caracteres **que pode conter** espaços, use gets (lembrando que \n é consumido do buffer) :

  ```
  char palavra[50];
  gets(palavra);
  ```

Entretanto **fique ciente** de que na prática profissional é melhor usar o fgets, pois não há perigo de buffer overflow.

O \n deixado no buffer não é problema quando nas leituras seguintes o scanf ignorar o \n. Por exemplo, no código a seguir não há problema pois o scanf seguinte ao scanf da string ignora o \n que ficou no buffer.

```
char palavra[50];
int n;
scanf("%s", palavra);
scanf("%d", &n);
```

Mas, como vimos, gera problema no código a seguir pois o especificador de formato %c do scanf não ignora o \n que ficou no buffer.

```
char palavra[50];
char t;
scanf("%s", palavra);
scanf("%c", &t);
```