Leitura de Strings em `C`

Daniel Saad Nogueira Nunes

O método mais simples é utilizar o `scanf` da maneira mais crua:

```
scanf("%s",str);
```

Este método tem algumas limitações:

- Ele lê a entrada até o próximo caractere "*whitespace*" (tabulação, nova linha ou espaço).
- Se mais caracteres do que o tamanho da string forem digitadas, teremos um **buffer overflow**.

Se quisermos ler a linha completa, uma possibilidade é usar o `gets`:

```
gets(str);
```

No entanto o gets compartilha do problema do `scanf`:

- Se a linha tiver mais caracteres do que a string comporta teremos um **buffer overflow**.

Uma maneira de utilizar o `scanf` simulando o `gets` (inclusive com os mesmos problemas) é fazer:

```
scanf("%[^\n]",str);
```

Aqui ele diz para ler o buffer de entrada até encontrar o primeiro '\n'. No entanto, ele não descarta o caractere de nova linha do buffer, como o `gets` faz, então é necessário modificá-lo para:

```
scanf("%[^\n]%*c",str);
```

Que descarta o caractere de nova linha.

Tanto o `gets` como o `scanf` simples ou modificado possuem o problema do **buffer overflow**. Uma forma de contornar isto, é especificar para o `scanf`, quantos caracteres no máximo devem ser lidos. Então, supondo que o tamanho máximo da *string* é 80 caracteres, podemos fazer:

```
scanf("%79[^\n]%*c",str);
```

A linha acima diz que queremos ler no máximo 79 caracteres ou até encontrar o fim da linha e descartamos o caractere de nova linha. Colocamos 79, pois devemos reservar um espaço para o '\0'.

Outra função útil é a função `fgets`.

```
fgets(str,80,stdin);
```

A função `fgets` lê uma linha de um arquivo, mas em `C`, o teclado (entrada padrão), também é considerado um arquivo de nome `stdin`. Além de ler a linha, ela certifica-se que ela não irá ler mais do que 79 caracteres da entrada (reservando espaço para o '\0').

Um problema do `fgets` é que ele também armazena o caractere de nova linha na string, sendo necessário sobrescrevê-lo se não for interessante tê-lo na string.