Leitura Segura de Strings

As funções da biblioteca padrão de C que podem ser usadas para leitura de *strings* não são suficientemente adequadas para esta finalidade básica. Nesta seção, serão descritos alguns problemas apresentados pelas funções da biblioteca padrão de C utilizadas na leitura de *strings*. Em seguida, será proposta uma função com esta finalidade que supera os problemas apontados. Finalmente, será apresentado um programa que demonstra o uso da função que será especificada e implementada aqui.

Problemas com as Funções de Leitura de Strings da Biblioteca Padrão

Serão apresentados a seguir alguns problemas que tornam as funções da biblioteca padrão de C freqüentemente utilizadas na leitura de *strings* inadequadas para esta finalidade.

Função scanf()

O problema mais sério que ocorre com a função **scanf()** é que ela simplesmente não é capaz de ler *strings* contendo espaços em branco.

Função gets()

A função **gets()** é provavelmente a função mais criticada de todas as funções da biblioteca padrão de C e aconselha-se radicalmente nunca utilizá-la. O problema mais grave com **gets()** é que ela pode corromper memória quando o usuário digita mais caracteres do que o array passado como argumento pode comportar. Lembre-se que não se devem fazer suposições otimistas sobre o nível cognitivo dos usuários ou suas intenções.

Função fgets()

A função **fgets()** é bem mais segura do que **gets()**, já que se pode especificar o número máximo de caracteres que podem ser lido no meio de entrada. O problema com a função **fgets()** é que ela inclui o caractere `'\n'` (que representa o pressionamento da tecla `[ENTER]` ou `[RETURN]`) e, muito provavelmente, quando se solicita a introdução de um *string* no meio de entrada padrão, não se está interessado em processar este caractere.

Algoritmo para Leitura de Strings

A função de leitura de *strings*, cuja implementação será apresentada em seguida, possui dois argumentos:

- `string`: este argumento de saída é um array de caracteres que receberá o *string* lido.
- `nElementos`: argumento de entrada que informa o tamanho (i.e., número de elementos) do array `string`.

A função de leitura segue o seguinte algoritmo:

1. Enquanto não faltar apenas um caractere para completar o array `string` ou um caractere de quebra de linha `'\n'` não for encontrado, faça o seguinte:
    1.1 Leia um caractere do meio de entrada padrão
    1.2 Se o caractere não for um caractere de quebra de linha, coloque-o no array (`string`)
2. Coloque o caractere terminal de *string* `'\0'` no array `string`
3. Verifique se o *buffer* de entrada foi esvaziado:
    3.1 Se o *buffer* de entrada estiver vazio ou contiver apenas o caractere `'\n'`, retorne um ponteiro para o *string* lido.
    3.2 Se o *buffer* de entrada não estiver vazio e contiver caracteres além de `'\n'`, retorne o ponteiro nulo indicando que o usuário digitou um número de caracteres maior do que o array podia suportar.

Note que uma das condições de parada do laço do passo 1 é faltar um caractere para completar o array e não o fato de este array estar completo. Isto ocorre porque é necessário deixar espaço para completar o array com o caractere terminal de *string* `'\0'`.

Implementação da Função de Leitura de Strings

A função `LeString()` apresentada a seguir tenta ler *strings* no meio de entrada padrão usando **getchar()** e retorna o *string* lido apenas quando o número de caracteres lidos estiver dentro do número máximo de caracteres permitido. (As linhas de interesse da função foram numeradas para facilitar a discussão que segue a apresentação da função.)

]

```
   char *LeString(char * string, unsigned nElementos)
   {
      unsigned contador = 0; /* Conta o número de caracteres lidos */
      char     caractereLido;

1.    caractereLido = getchar();
2.    while ((caractereLido != '\n') && (contador < nElementos-1)) {
3.       string[contador++] = caractereLido;
4.       caractereLido = getchar();
      }

5.    string[contador] = '\0'; /* Termina o string */

6.    if ((caractereLido == '\n') || (getchar() == '\n')) // Tudo bem
7.       return string;
8.    else {      /* Usuário digitou caracteres demais */
9.       LimpaBuffer();
         return (char *) 0;
      }
   }
```

Comentários sobre a função LeString()

1. Esta instrução simplesmente lê o primeiro caractere.

2. Esse laço **while** será responsável pela leitura dos caracteres restantes e o subseqüente armazenamento dos mesmos no array `string`. Existem duas condições de parada deste laço: (1) ter encontrado o caractere `'\n'` ou (2) ter lido no máximo `nElementos - 1` caracteres.

3. Aqui, é feito o armazenamento do caractere lido no array `string` e, ao mesmo tempo, incrementa-se a variável `contador`. É essencial que este incremento seja sufixo.

4. Simplesmente, lê o próximo caractere.

5. Neste ponto, a variável `contador` contém o índice do caractere que segue o último caractere armazenado no array `string`. Isto ocorre porque o operador sufixo de incremento foi utilizado. Esta instrução coloca o caractere terminal de *string* `'\0'` na posição indicada pela variável `contador`.

6. A instrução **if** testa o último caractere lido (armazenado na variável `caractereLido`) e o próximo caractere [obtido com **getchar()**] com `'\n'`.

7. Se o último caractere lido for `'\n'`, o usuário digitou menos caracteres do que o máximo permitido. Se o próximo caractere no *buffer* de entrada for `'\n'`, o usuário digitou exatamente o máximo permitido. Nestes dois casos, tudo ocorreu normalmente e resta apenas retornar o endereço do array `string`.

8. Se nem o último caractere lido nem o próximo caractere no *buffer* de entrada for `'\n'`, o usuário digitou caracteres além do permitido.

9. Nesse caso, limpa-se o *buffer* de entrada e retorna-se um ponteiro nulo.

Limpeza do Buffer de Entrada

A função `LimpaBuffer()` apresentada a seguir faz a faxina no buffer de entrada padrão. Isto é, a função `LimpaBuffer()` lê e descarta todos os caracteres que porventura tenham sido deixados no buffer de entrada em alguma tentativa de leitura de dados. Uma limitação desta função é que, se não houver pelo menos um caractere `'\n'` no buffer de entrada, ela irá aguardar a introdução deste caractere antes de liberar a entrada.

```
void LimpaBuffer(void)
{
    int valorLido = getchar(); /* valorLido deve ser int! */

         /* Lê caracteres no buffer de entrada enquanto     */
         /* não encontra final de linha ou final de arquivo */
    while ((valorLido != '\n') && (valorLido != EOF))
        valorLido = getchar();}
}
```

Comentários sobre a função LimpaBuffer()

Esta função pode ser implementada de modos diferentes.

A variável local `valorLido`, que armazena os caracteres deixados no *buffer*, deve ser do tipo **int** (e não do tipo **char**) porque existe a possibilidade de **getchar()** retornar um valor (**EOF**) que indica tentativa de leitura além do final de arquivo e este valor é do tipo **int**. O laço **while** simplesmente lê todos os caracteres encontrados no buffer de entrada padrão e pára quando encontra o caractere `'\n'` (representando `[ENTER]` digitado pelo usuário) ou quando **getchar()** retorna **EOF**.

Uso da Função de Leitura

A função **main()** a seguir aceita qualquer *string* que respeite o número máximo de caracteres permitido e encerra apenas quando um *string* dentro desse limite for lido.

```
   int main(void)
   {
1.    char strEntrada[TAMANHO_STRING];
2.    char *stringLido;

      printf("\nIntroduza um string com no maximo %d caracteres: ",
             TAMANHO_STRING - 1);
3.    stringLido = LeString(strEntrada, TAMANHO_STRING);

4.    while(1) {
5.       if (stringLido) {
            printf("\nO string lido foi: %s\n", stringLido);
            break;    /* Tchau while */
         }

6.       printf("\nO string introduzido tem mais de %d caracteres: ",
                 TAMANHO_STRING - 1);
         printf("\nDigite um string com no maximo %d caracteres: ",
                 TAMANHO_STRING - 1);
         stringLido = LeString(strEntrada, TAMANHO_STRING);
      }

      return 0;
   }
```

Comentários sobre a função main()

1. O array `strEntrada` armazena os *strings* lidos e `TAMANHO_STRING` é uma macro (constante simbólica) definida no arquivo que contém a função **main()**. Esta constante representa o número de elementos do array `strEntrada`, que irá conter o *string* introduzido pelo usuário.

2. O ponteiro `stringLido` apontará para o *string* lido se este for válido ou conterá um valor nulo em caso contrário.

3. Esta instrução tenta ler um *string* usando a função `LeString()`.

4. Esse laço termina com uma instrução **break** em seu corpo que será executada apenas quando um *string* válido for lido.

5. Esta instrução **if** testa se o *string* foi lido sem problemas; isto é, se a variável `stringLido` contém um valor diferente de **NULL**. Se este for o caso, imprime-se o *string* lido e sai-se do laço.

6. Se a instrução **break** não foi executada, o *string* não foi lido corretamente (i.e., a variável `stringLido` é nula). As instruções seguintes fazem uma nova tentativa de leitura.