Linguagem de Programação C

Strings

Cristiano Lehrer

http://www.ybadoo.com.br/

Introdução

- Uma *string* é um conjunto de caracteres armazenados num vetor:
  - Como em C uma *string* não é um tipo básico, a única forma de representar um conjunto de caracteres é recorrendo a um vetor de caracteres.
- Uma *string* é apenas um vetor de caracteres:
  - O inverso é falso, ou seja, um vetor de caracteres pode não ser uma *string*.

Exemplos de String

String "Zé"

Z	é	\0	...	...	...
v[0]	v[1]	v[2]	v[3]	v[4]	v[5]

String "Luiz"

L	u	i	z	\0	...
v[0]	v[1]	v[2]	v[3]	v[4]	v[5]

String "" (string vazia)

\0	...	...	...	...	...
v[0]	v[1]	v[2]	v[3]	v[4]	v[5]

Delimitador de String

- O caractere ASCII selecionado como delimitador de uma *string* é o caractere cujo código ASCII é igual a ZERO, isto é, o caractere número zero e que se representa por `'\0'`:
  - O caractere `'\0'` não possui nenhuma representação gráfica.
  - O caractere `'\0'` cujo código ASCII é 0 nada tem a ver com o caractere `'0'`, um outro caractere totalmente diferente (código ASCII 48).
  - O caractere `'\0'` deverá ser sempre o último caractere de uma *string*.

Carga Inicial Automática de Strings

```
char nome[20] = "Fulano";
```

F	u	l	a	n	o	\0	...	...
nome[0]	nome[1]	nome[2]	nome[3]	nome[4]	nome[5]	nome[6]	...	nome[19]

```
char nome[20] = {'F', 'u', 'l', 'a', 'n', 'o'};
```

F	u	l	a	n	o	\0	...	...
nome[0]	nome[1]	nome[2]	nome[3]	nome[4]	nome[5]	nome[6]	...	nome[19]

```
char nome[] = "Fulano";
```

F	u	l	a	n	o	\0
nome[0]	nome[1]	nome[2]	nome[3]	nome[4]	nome[5]	nome[6]

```
char *nome = "Fulano";
```

F	u	l	a	n	o	\0
nome[0]	nome[1]	nome[2]	nome[3]	nome[4]	nome[5]	nome[6]

Vetores de Caracteres versus String

- Vetor de caracteres:

```
char vogais[] = {'a', 'e', 'i', 'o', 'u'};
```

a	e	i	o	u
vogais[0]	vogais[1]	vogais[2]	vogais[3]	vogais[4]

- String:

```
char vogais[6] = {'a', 'e', 'i', 'o', 'u'};
```

a	e	i	o	u	\0
vogais[0]	vogais[1]	vogais[2]	vogais[3]	vogais[4]	vogais[5]

```
char vogais[] = "aeiou";
```

a	e	i	o	u	\0
vogais[0]	vogais[1]	vogais[2]	vogais[3]	vogais[4]	vogais[5]

Função printf

- Pode receber uma constante:

```
printf("Hello World");
```

- Pode receber uma variável:

```
char nomeProprio[100] = "Fulano";
char sobrenome[100] = "Silva";
printf("Nome: %s, %s\n", sobrenome, nomeProprio);
```

- O formato `%s` indica para o método que se trata de uma *string*.

Função puts

- Permite unicamente a escrita de *strings*, sejam constantes ou armazenadas em variáveis:
  - A única diferença entre o método `printf`, é que o método `puts` coloca automaticamente uma quebra de linha no final da *string*:
  - São instruções equivalentes:

```
puts("Hello World");

printf("Hello World\n");
```

Função scanf

- Permite a leitura de *strings* utilizando o formato `%s`:
- A variável que receberá a *string* não deverá ser precedido de um `&`.
- A função realiza apenas a leitura de uma única palavra.
- Lê todos os caracteres até encontrar um <ESPAÇO>, <TAB> ou <ENTER>

```
char nome[50];

printf("Forneça o seu nome: ");

scanf("%s", nome);

printf("Seu nome é %s\n", nome);
```

Função gets

- Permite colocar, na variável que recebe por parâmetro, todos os caracteres introduzidos pelo usuário:

```
char nome[50];

printf("Forneça o seu nome: ");

gets(nome);

printf("Seu nome é %s\n", nome);
```

Função fgets

- Permite colocar, na variável que recebe por parâmetro, todos os caracteres introduzidos pelo usuário, até o limite especificado pelo programador:

```
char nome[50];

printf("Forneça o seu nome: ");

fgets(nome, 50, stdin);

printf("Seu nome é %s\n", nome);
```

Deveria ser a função base para a leitura de strings em todos os programas escritos em C!

Passagem de Strings para Funções (1/2)

- A passagem de *strings* para funções é exatamente igual à passagem de vetores para funções, uma vez que qualquer *string* é sempre um vetor de caracteres.

```c
/* string length */
int strlen2(char s[], int max)
{
  int i = 0;

  while((s[i] != '\0') && (i < max))
  {
    i = i + 1;
  }

  return i;
}
```

```c
int main()
{
  char palavra[100] = "Fulano";

  int tam = strlen2(palavra, 100);

  printf("\"Fulano\" possui %d caracteres.\n", tam);

  return 0;
}
```

Passagem de Strings para Funções (2/2)

- No caso de *string*, não é necessário passar o tamanho do *array*, pois o mesmo é obtido através do caractere `'\0'`.

```c
/* string length */
int strlen2(char s[])
{
  int i = 0;

  while(s[i] != '\0')
  {
    i = i + 1;
  }

  return i;
}
```

```c
int main()
{
  char palavra[100] = "Fulano";

  int tam = strlen2(palavra);

  printf("\"Fulano\" possui %d caracteres.\n", tam);

  return 0;
}
```

Questões de Segurança (1/3)

- Inimigo público nº 1: **estouro de buffer**:
  - Práticas ruins de codificação.
  - O fato do C dar aos programadores muitas maneiras de atirar no próprio pé.
  - Falta de funções de tratamento de *strings* seguros e fáceis de usar.
  - Falta de validação dos limites de um *array*.
  - Ignorância sobre as consequências reais dos erros.

Questões de Segurança (2/3)

- O *Microsoft Security Response Center* estima o custo da emissão de um boletim de segurança e o *patch* associado em US$ 100.000,00 e isso é só o começo:
  - Horas dos profissionais que irão aplicar os *patch*.
  - Horas dos profissionais que irão identificar os computadores sem o *patch* instalado.
  - O risco de estar vulnerável até a aplicação do *patch*.

Questões de Segurança (3/3)

- Um estouro de *buffer* é explorável, ou seja, usuários experientes podem utilizar o estouro de *buffer* para alterar o comportamento do programa, fazendo com que o mesmo realize as funções por eles designadas, ao invés das originalmente projetadas:
  - Vírus de computador.
  - Cavalos de tróia.
- Por isso:

SEMPRE UTILIZE FUNÇÕES SEGURAS

- No caso de *strings*, funções que recebam junto o tamanho máximo permitido para a operação.