Aula: Array de caracteres e *strings*

Introdução a Programação

Túlio Toffolo & Puca Huachi
http://www.toffolo.com.br

Departamento de Computação
Universidade Federal de Ouro Preto

Aula: Array de caracteres e *strings*

Aula: Array de caracteres e *strings*

Cadeias de caracteres

- Variáveis do tipo `char` são usadas para armazenar um caracter (tamanho = 1 byte).
- Em C uma cadeia de caracteres (***string***) é implementada como um **vetor** do tipo `char`.

Cadeias de caracteres

Lembrando que...

- Caracteres literais são representados por aspas simples:

```
1 char c1 = 'a';
2 char c2 = 'A';
```

- Variáveis do tipo `char` recebem valores literais do tipo caractere.
  - Que representam o caractere correspondente, conforme o sistema de codificação adotado.
  - Podemos converter um caractere para um número e vice-versa.
  - **Lembram da tabela ASCII?**

Tabela ASCII

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Exemplo

Crie um programa em C/C++ que imprime o código (em decimal) relativo a um caractere digitado pelo usuário.

```c
#include <stdio.h>

int main()
{
    char c;
    printf("Digite um caractere: ");
    scanf("%c", &c);
    printf("Código (decimal) de '%c' é  %d\n", c, (int) c);
    return 0;
}
```

Cadeia de caracteres

Cadeias de caracteres são simplesmente arrays/vetores de caracteres que terminam com o caractere '\0':

- O caractere especial '\0' indica o final da cadeia de caracteres
- Note que para armazenar 10 caracteres precisamos de 11 posições
  - Uma posição adicional para o caractere '\0'
- Estas cadeias são também chamadas de ***strings***

Exemplos

Suponha um array de 15 caracteres

- `char nome[15];`

?	?	?	?	?	?	?	?	?	?	?	?	?	?	?

- `scanf("%s", nome);`
  (suponha que o usuário digitou Puca)

P	u	c	a	\0	?	?	?	?	?	?	?	?	?	?

Operações em cadeia de caracteres

A função `strlen()` (abreviação de *string length*) é utilizada para calcular o tamanho de uma *string*.

- Sim, **calcular** o tamanho: a função percorrerá o array de caracteres em busca do caractere '\0'.
- Lembre-se, portanto, que há um custo elevado em chamar essa função várias vezes.

Operações em cadeia de caracteres

Exemplo:

```
1  char nome[15];
2  scanf("%s", nome); // le uma 'string' do usuário
3
4  int tamanho = strlen(nome); // calcula o tamanho da 'string'
5  printf("%d\n", tamanho);
```

Se o usuário digitar 'Toffolo', qual será o conteúdo do vetor?

T	o	f	f	o	l	o	\0	?	?	?	?	?	?	?

O tamanho da cadeia de caracteres (ou seja, a 'posição' do \0):

```
1  7
```

Inicialização de *strings*

Uma cadeia de caracteres (*string*) pode ser inicializada facilmente:

```
1  char nome[] = "Toffolo";
```

- Ao utilizar a construção acima, o array `nome` terá um total de 8 posições, pois o '\0' é automaticamente inserido no final do vetor.

0	1	2	3	4	5	6	7
T	o	f	f	o	l	o	\0

- Note que o tamanho da *string* é 7 (ou seja, a 'posição' do `\0`)

Inicialização de *strings*

Por outro lado, o código abaixo inicializa um array de 15 caracteres:

```
1  char nome[15] = "Toffolo";
```

- O array terá tamanho 15
- Note que o '\0' será inserido no vetor:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
T	o	f	f	o	l	o	\0	?	?	?	?	?	?	?

Exemplo

Implemente a função `strlen()` com o seguinte protótipo:

```
1  int strlen(char str[]);
```

Aula: Array de caracteres e *strings*

Biblioteca <string.h>

A biblioteca <string.h> contém algumas funções úteis:

- `char *strcpy(char x[], char y[]);`
  copia a string `x` (inclusive o byte \0 final) no espaço alocado para `y`. Cabe ao usuário garantir que o espaço alocado a `y` tem pelo menos `strlen(x) + 1` bytes. Retorna o endereço de `x`.
- `char *strcat(char x[], char y[]);`
  concatena as strings `x` e `y`, isto é, acrescenta `y` ao final de `x`. Retorna o endereço da string resultante, ou seja, o endereço de `x`. Cabe ao usuário garantir que o espaço alocado a `x` é suficiente para comportar `strlen(y)` bytes adicionais.
- `int strcmp(char x[], char y[]);`
  compara lexicograficamente as strings `x` e `y`. Retorna um número estritamente negativo se `x` vem antes de `y`, 0 se `x` é igual a `y` e um número estritamente positivo se `x` vem depois de `y`.

Alguns comentários...

O que o código a seguir imprimirá?

```c
#include <stdio.h>

int main() {
    char nome1[] = "Toffolo";
    char nome2[] = "Toffolo";

    if (nome1 == nome2)
        printf("Nomes iguais (operador ==).\n");
    else
        printf("Uai... nome1 e nome2 não são iguais (operador ==).\n");

    return 0;
}
```

Fácil, né...

```
Uai... nome1 e nome2 não são iguais (operador ==).
```

Alguns comentários...

O que o código a seguir imprimirá?

```c
#include <stdio.h>

int main() {
    char nome1[] = "Toffolo";
    char nome2[] = "Toffolo";

    if (nome1 == nome2)
        printf("Nomes iguais (operador ==).\n");
    else
        printf("Uai... nome1 e nome2 não são iguais (operador ==).\n");

    return 0;
}
```

- Lembrem-se que `nome1` e `nome2` são vetores/arrays!
- Assim, os valores de `nome1` e `nome2` são **endereços de memória**!

Alguns comentários...

O correto é utilizar a função `strcmp`:

```c
#include <stdio.h>
#include <string.h>

int main() {
    char nome1[] = "Toffolo";
    char nome2[] = "Toffolo";

    if (strcmp(nome1, nome2) == 0)
        printf("Nomes iguais!\n");
    else
        printf("Uai... nome1 e nome2 não são iguais.\n");

    return 0;
}
```

- `strcmp` retornará um número negativo se `nome1` < `nome2`
- `strcmp` retornará **zero** se `nome1` for igual a `nome2`
- `strcmp` retornará um número positivo se `nome1` > `nome2`

Aula: Array de caracteres e *strings*

Alguns comentários...

Observe o exemplo a seguir...

```c
#include <stdio.h>

int main() {
    char nome[] = "Tulio";

    printf("Endereço de nome = %p\n", nome);
    printf("A string em nome = %s\n", nome);

    return 0;
}
```

- Um exemplo de saída:

```
Endereço de nome = 0x7ffeedd4c326
A string em nome = Tulio
```

Alguns comentários...

Ao utilizar o formato `%s`, o seguinte acontece:

1. O conteúdo do endereço de memória apontado pelo vetor é acessado.
2. Se o conteúdo for o caractere '`\0`', o algoritmo termina.
3. Caso contrário, o caractere é impresso e o passo **2** é repetido considerando o próximo endereço de memória (o próximo byte).

Percebem o risco?

- E se não existir um caractere '`\0`' no vetor?
- O algoritmo continuará imprimindo caracteres até encontrar algum '`\0`' ou falhar...
- A boa notícia é que... '`\0`' **corresponde ao byte 0**.

Manipulação de *strings*

Podemos utilizar `scanf` para ler *strings*.

- No entanto, `scanf` finalizará a leitura quando encontrar um espaço.
- Exemplo:

```
1 char nomeCompleto[100];
2 printf("Digite o nome completo: ");
3 scanf("%s", nomeCompleto);
4 printf("Nome: %s", nomeCompleto);
```

- Exemplo de execução do código acima:

```
1 Digite o nome completo: Tulio Angelo Machado Toffolo
2
3 Nome: Tulio
```

Manipulação de *strings*

Uma alternativa é utilizar a função `gets` ou `fgets`:

- `gets(x)`: lerá da entrada o que for digitado pelo usuário até uma quebra de linha (‘\n’) ser detectada e armazenará na *string* `x`.
- `fgets(x, n, stdin)`: lerá da entrada o que for digitado pelo usuário até uma quebra de linha (‘\n’) ser detectada ou o limite máximo de `n` caracteres ser atingido, e armazenará na *string* `x`.
  - `stdin`: constante que indica a entrada padrão (entrada do usuário). É uma abreviação para *standard input*.

Importante: `fgets` **incluirá o caractere** ‘**\n**’ em `x`.

Manipulação de *strings*

Atenção: evite utilizar `gets` (função *deprecated*).

Mas... Porquê?

- Porque não sabemos quantos caracteres o usuário irá digitar...
- E podemos gerar inúmeros problemas por conta disso...
- Tente colocar 20 caracteres em um `char nome[10];`

Aula: Array de caracteres e *strings*

Parâmetros da função main

Até aqui, utilizamos a função `main` sem parâmetros:

```
1  int main()
2  {
3      ...
4
5      return 0;
6  }
```

Mas a função `main` também pode receber parâmetros:

- `int argc`: número de argumentos
- `char **argv`: valor dos argumentos
  - poderia ser, alternativamente, `char *argv[]`

Parâmetros da função main

```c
#include <stdio.h>

int main(int argc, char *argv[])
{
    for (int i = 0; i < argc; i++)
        printf("argumento %d = %s\n", i, argv[i]);

    return 0;
}
```

O que o código acima irá imprimir?

Parâmetros da função main

O fato é que `argv` é um vetor de vetores de caracteres!

(ou um array de arrays de caracteres)

- Mas um vetor de caracteres pode ser visto como uma string...
- Portanto, `argv` é na prática um vetor de strings :)

Aula: Array de caracteres e *strings*

Exercícios

Exercício 1

Crie um programa que verifica se a palavra “UFOP” (com letras maiúsculas) foi passada por argumento na linha de comando. Se sim, o programa deve imprimir:

```
1 Bem vindo a Ouro Preto!
```

Caso contrário, nada deve ser impresso pelo programa.

Perguntas?