Introdução à Programação C

Fabio Mascarenhas - 2014.2

http://www.dcc.ufrj.br/~fabiom/introc

Vetores

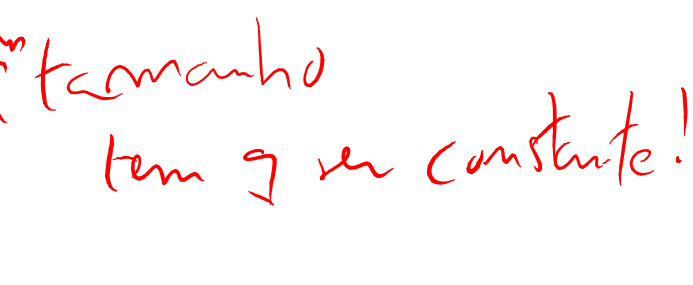
- Para vários problemas precisamos de uma maneira de representar uma *sequência* de valores com um número de elementos arbitrário
  - Por exemplo, o jogo da forca poderia usar uma sequência com quaisquer número de letras
- Em C, uma maneira de representar sequências de valores é com *vetores*
- Um vetor é uma sequência de elementos em que cada elemento tem um *índice* associado, de 0 até um a menos que o número de elementos
  - Um vetor de 5 elementos tem índices 0, 1, 2, 3 e 4

Declarando vetores

- A declaração de um vetor depende do local onde estamos declarando

- Se for uma variável local ou global, declaramos um vetor adicionando um par de colchetes ao nome da variável, com o número de elementos do vetor dentro dos colchetes

```
int xs[10];
char palavra[5];
double ys[10];
```



- Se for um *parâmetro* declaramos um vetor adicionando apenas o par de colchetes ao nome, sem um número de elementos; devemos usar outro parâmetro para indicar para a função quantos elementos o vetor tem!

```
(char palavra[], int npalavras)
```

Inicialização de um vetor

- Se declaramos uma variável vetor e não a inicializamos o vetor tem valores arbitrários

- Podemos inicializar o vetor dando uma *lista* de valores entre chaves após a declaração:

```
char palavra[5] = { 'F', 'O', 'R', 'C', 'A' };
```

- Esses valores têm que ser *constantes*

- Quando inicializamos um vetor o tamanho entre os colchetes é opcional; se omitimos ele o tamanho é dado pelo número de valores na lista de inicialização

```
char palavra[] = { 'F', 'O', 'R', 'C', 'A' };
```

Indexação

- Podemos ler elementos do vetor e escrever elementos no vetor com operações de *indexação*

- Na indexação usamos o nome do vetor seguido de colchetes, com o *índice* que queremos ler ou escrever dentro dos colchetes: `palavra[3]`, `xs[0]`

- O índice não precisa ser uma constante, pode ser qualquer expressão inteira: `palavra[i]`, `ys[xs[0]]`

- Uma indexação usada em uma expressão lê um elemento do vetor, e uma indexação usada no lado esquerdo de uma atribuição escreve um elemento no vetor

```
ys(xs(0)) = 10.5;
```

Exemplo: inicialização

- As duas formas de declarar e inicializar o vetor palavra abaixo são equivalentes (produzem o mesmo vetor):

```
char palavra[5];
palavra[0] = 'F';
palavra[1] = 'O';
palavra[2] = 'R';
palavra[3] = 'C';
palavra[4] = 'A';
```

```
palavra[0] = getch();
palavra[1] = getch();
palavra[2] = getch();
palavra[3] = getch();
palavra[4] = getch();
```

```
char palavra[] = { 'F', 'O', 'R', 'C', 'A' };
```

Vetores como parâmetros

- Vetores declarados como parâmetros não têm tamanho, portanto o número de elementos deve ser passado em outro parâmetro, mas isso não é a única diferença

- Com um parâmetro normal, atribuir o parâmetro **não afeta** o argumento; a função a seguir não funciona:

```
void troca(int a, int b) {
        int c = a;
        a = b;
        b = c;
}
```

- Com um vetor, escrever no vetor **afeta** o vetor passado como argumento, e a função a seguir troca os valores do primeiro e segundo elementos do vetor:

```
void troca(int v[]) {
        int x = v[0];
        v[0] = v[1];
        v[1] = x;
}
```

Jogo da forca com vetores

- Vamos modificar nossa aplicação de forca para usar um vetores de quatro elementos para a palavra e os acertos, ao invés de quatro variáveis diferentes

Strings

- Strings, ou cadeias de caracteres, são vetores de caracteres em C
- Cada caractere é um valor do tipo `char`, com um código numérico entre 0 e 127
- O caractere de código 0 é especial, e indica o final da string
- Funções que trabalham com strings não precisam saber o tamanho da string, pois usam o caractere 0 para saber quando chegou no final

Códigos de caracteres

	0	1	2	3	4	5	6	7	8	9
30			sp	!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	S	t	u	v	w
120	x	y	z	{	|	}	~			

Inicialização

- Quando criamos uma string com aspas duplas, o caractere 0 é inserido no final implicitamente:

```
char palavra[] = "Forca";
```

- A declaração acima cria um vetor com 6 caracteres, e é equivalente à declaração abaixo:

```
char palavra[] = { 'F', 'o', 'r', 'c', 'a', 0 };
```

`printf` e `scanf`

- Podemos imprimir uma string usando com `printf`, com o código de formato `%s`:

```
printf("A palavra e %s.", palavra);
```

- Também podemos pedir uma string ao usuário e armazenar ela em um vetor já existente, mas precisamos garantir que o vetor tem espaço suficiente para o que o usuário vai entrar:

```
char palavra[81];
scanf("%80[^\n]", palavra);
```

- Note que não usamos & antes de palavra! O código de formato usado lê os primeiros oitenta caracteres digitados pelo usuário, ou até a quebra de linha

Manipulando strings

- Como uma string é um vetor, podemos indexá-las para ler caracteres da string, ou substituir caracteres:

```
char palavra[] = "Forca";
palavra[3] = 'r';
printf("%s", palavra); /* imprime Forra */
```

- A biblioteca string.h também tem algumas funções úteis para manipular strings:
  - `strlen` dá o número de caracteres da string (sem contar o 0 no final!)
  - `strcpy` copia os caracteres de uma string *por cima dos caracteres* de outra, `strcat` *adiciona* os caracteres de uma string aos de outra
  - `strcmp` compara duas strings

strlen

- A função `strlen` recebe uma string e retorna quantos caraceteres ela tem, ignorando o 0 no final:

```
char palavra[81];
scanf("%80[^\n]", palavra);
printf("Palavra tem %d caracteres\n", strlen(palavra));
```

- Ela é útil para garantir que temos espaço suficiente para copiar os caracteres de uma string para outra quando usamos `strcpy` ou `strcat`

strcpy e strcat

- A função strcpy substitui os caracteres de uma string pelos caracteres de outra:

```
char palavra[81];
scanf("%80[^\n]", palavra);
strcpy(palavra, "Forca");
printf("%s", palavra); /* imprime Forca */
```

- A função strcat adiciona os caracteres de uma string aos de outra:

```
char palavra[81];
strcpy(palavra, "Forca");
strcat(palavra, "Forca");
printf("%s", palavra); /* imprime ForcaForca */
```

- Em ambos os casos a string destino precisa ter espaço suficiente!

`strcmp`

- A função `strcmp` compara duas strings caractere a caractere, do primeiro ao último, e:
  - Retorna 0 se todos os caracteres são iguais
  - Retorna -1 se encontrou um caractere na primeira string que é menor que seu correspondente na segunda
  - Retorna 1 se encontrou um caractere na primeira string que é maior que seu correspondente na segunda
- Essa comparação é chamada *lexicográfica*, lembrando que caracteres minúsculos são *maiores* que maiúsculos

Conversão entre strings e números

- A função `sprintf` constrói uma string a partir de um modelo com códigos de formato e valores, do mesmo modo que `printf`, só que copia os caracteres dessa string para uma string passada ao invés de imprimir no console:

```
char frase[81];
sprintf(frase, "%s pagou %.2lf para %s", "Joao", 2.5, "Maria");
/* frase agora é "Joao pagou 2.50 para Maria" */
```

- Devemos garantir que a string tem espaço para receber todos os caracteres!

- `sprintf` é uma forma conveniente de converter números para strings

- Para fazer o contrário temos as funções `atoi` e `atof`, que convertem strings para inteiros e doubles, respectivamente

Strings e score

- Como exemplo de uso de algumas funções para strings, vamos adicionar duas funcionalidades pro nosso jogo de forca:
  - Na parte inferior da tela vamos mostrar todas as letras que o jogador já tentou e que não fazem parte da palavra
  - Vamos também mostrar um score, que começa em 100 e abaixa 20 pontos para cada erro que o jogador comete