

Vetores, *strings* e matrizes

- Vetores

- Série de variáveis do mesmo tipo em endereços contíguos;
- Evitar declaração de várias variáveis.
- *tipo nome [numero_de_elementos]*

Vetores, *strings* e matrizes

- Vetores

- Exemplo: *int vetor_novo[5];*
- O número de elementos deve ser um valor constante.



Vetores, *strings* e matrizes

- Vetores – inicialização:

- *int vetor_novo[5] = {3, 4, 78, 678, 20};*
- *int vetor_novo[] = {3, 4, 78, 678, 20};*

	0	1	2	3	4
vetor_novo	3	4	78	678	20
	int	int	int	int	int

Vetores, *strings* e matrizes

- Vetores – inicialização e acesso a elementos:

```
#include <stdio.h>

void main()
{
    int v[5] = {7, 3, 32, 45, 6};

    printf("%d %d %d %d %d",
        v[0], v[1], v[2], v[3], v[4]);
}
```

7 3 32 45 6

Vetores, *strings* e matrizes

- Vetores – inicialização e acesso a elementos:

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

```
void main()
```

```
{
```

```
    int v[ ] = {7, 3, 32, 45, 6};
```

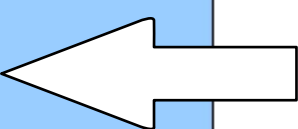
```
    v[0] = 315;
```

```
    v[3] = 723;
```

```
    printf("%d %d %d %d %d",
```

```
    v[0], v[1], v[2], v[3], v[4]);
```

```
}
```



0	1	2	3	4
7	3	32	45	6

Vetores, *strings* e matrizes

- Vetores – inicialização e acesso a elementos:

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

```
void main()
```

```
{
```

```
    int v[ ] = {7, 3, 32, 45, 6};
```

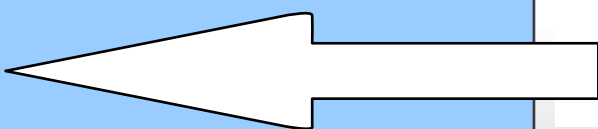
```
    v[0] = 315;
```

```
    v[3] = 723;
```

```
    printf("%d %d %d %d %d",
```

```
    v[0], v[1], v[2], v[3], v[4]);
```

```
}
```



	0	1	2	3	4
v	315	3	32	45	6

Vetores, *strings* e matrizes

- Vetores – inicialização e acesso a elementos:

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

```
void main()
```

```
{
```

```
    int v[ ] = {7, 3, 32, 45, 6};
```

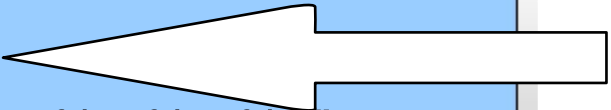
```
    v[0] = 315;
```

```
    v[3] = 723;
```

```
    printf("%d %d %d %d %d",
```

```
    v[0], v[1], v[2], v[3], v[4]);
```

```
}
```



	0	1	2	3	4
v	315	3	32	723	6

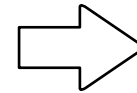
Vetores, *strings* e matrizes

- Vetores – inicialização e acesso a elementos:

```
#include <stdio.h>

void main()
{
    int v[ ] = {7, 3, 32, 45, 6};

    v[0] = 315;
    v[3] = 723;
    printf("%d %d %d %d %d",
        v[0], v[1], v[2], v[3], v[4]);
}
```



315 3 32 723 6

Vetores, *strings* e matrizes

● Vetores

- Nada impede o programador de acessar uma posição além daquelas definidas na declaração do vetor.

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

```
void main()
```

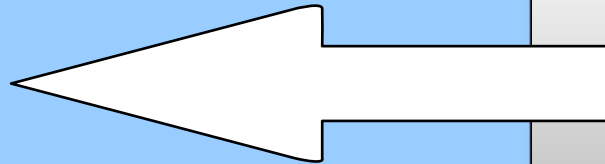
```
{
```

```
int v[ 5 ] = {7, 3, 32, 45, 6};
```

```
v[ 5 ] = 315;
```

```
v[ 10 ] = 723;
```

```
}
```



Não permita que
isso aconteça!
Não se pode prever
os erros decorrentes!!!

Vetores, *strings* e matrizes

- *Strings*

- Tabela ASCII: caracteres de texto são representados por valores hexadecimais (8 bits)

	0	1	2	3	4	5	6	7
0	NUL	DLE	space	0	@	P	`	p
1	SOH	DC1 XON	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3 XOFF	#	3	C	S	c	s
4	EOT	DC4	\$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(8	H	X	h	x
9	HT	EM)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[k	{
C	FF	FS	,	<	L	\	l	
D	CR	GS	-	=	M]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	del

Vetores, *strings* e matrizes

● *Strings*

- Para armazenar uma letra em ASCII, é necessário um byte: uma variável do tipo char.
- Para não armazenarmos várias letras de um texto em muitas variáveis diferentes, utilizamos um vetor de chars.
- Exemplo: `char palavra [5] = { 'T', 'e', 'x', 't', 'o' };`

	0	1	2	3	4
palavra	'T'	'e'	'x'	't'	'o'
	char	char	char	char	char

Vetores, *strings* e matrizes

- *Strings*

- Podemos armazenar palavras menores neste vetor. Para indicar o final da palavra, utiliza-se o caracter '\0'.

	0	1	2	3	4
palavra	'O'	'l'	'a'	'\0'	'o'

Vetores, *strings* e matrizes

● *Strings*

- Para simplificar, pode-se usar o seguinte método:
- `char palavra[] = { 'A', 'B', 'C', 'D', '\0' };`
- `char palavra[] = "ABCD";`
- O resultado é o mesmo em ambos:

	0	1	2	3	4
palavra	'A'	'B'	'C'	'D'	'\0'

Vetores, *strings* e matrizes

- *Strings*

```
#include <stdio.h>
void main()
{
    char v[3];

    v = "Oi";
    v[ ] = "Oi";
    v = { 'O', 'i', '\0' };
    v[ ] = { 'O', 'i', '\0' };
}
```

```
#include <stdio.h>
void main()
{
    char v[3];

    v[0] = 'O';
    v[1] = 'i';
    v[2] = '\0';
}
```

Vetores, *strings* e matrizes

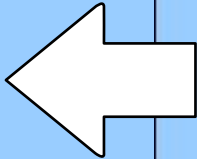
• *Strings*

- Ambos os métodos são válidos somente na inicialização do vetor.

```
#include <stdio.h>
void main()
{
    char v[3];

    v = "Oi";
    v[ ] = "Oi";
    v = { 'O', 'i', '\0' };
    v[ ] = { 'O', 'i', '\0' };
}
```

*Todos
errados!!!*



```
#include <stdio.h>
void main()
{
    char v[3];

    v[0] = 'O';
    v[1] = 'i';
    v[2] = '\0';
}
```

*Correto, pois
trabalha com
cada elemento
do vetor.*

Vetores, *strings* e matrizes

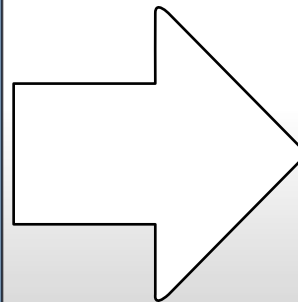
● *Strings* – Visualização

- A função printf() deve ser informada de que vai apresentar um caracter em ASCII através do símbolo %c:

```
#include <stdio.h>

void main()
{
    char v[ ] = "Oi";

    printf("%d %d %d",
        v[0], v[1], v[2]);
}
```



79 105 0

Vetores, *strings* e matrizes

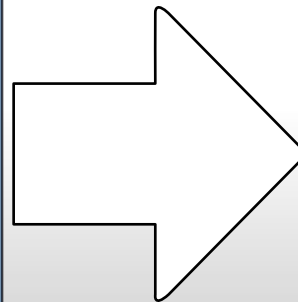
● *Strings* – Visualização

- A função printf() deve ser informada de que vai apresentar um caracter em ASCII através do símbolo %c:

```
#include <stdio.h>

void main()
{
    char v[ ] = "Oi";

    printf("%c %c %c",
           v[0], v[1], v[2]);
}
```



O i

Vetores, *strings* e matrizes

● *Strings* – Visualização

- Uma maneira mais simples do que acessar cada elemento do vetor é utilizar o símbolo %s:

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

```
void main()
```

```
{
```

```
    char v[ ] = "Oi";
```

```
    printf("%d %d %d\n",
```

```
           v[0], v[1], v[2]);
```

```
    printf("%c %c %c\n",
```

```
           v[0], v[1], v[2]);
```

```
    printf("%s", v);
```

```
}
```

79 105 0

O i

Oi

A palavra a ser visualizada
deverá terminar em '\0'

Vetores, *strings* e matrizes

- Matrizes

- Vetores de vetores - vetores multidimensionais;
- Útil para armazenar tabelas, *pixels* de imagens e videos etc.
- *tipo nome [num_elem_dim1] ... [num_elem_dimN]*

Vetores, *strings* e matrizes

- Matrizes

- Exemplo: `int vetor_novo[3][4];`

	0	1	2	3
vetor_novo 0				
1				
2				

- O número de elementos também deve ser um valor constante.

Vetores, *strings* e matrizes

- Matrizes - acesso a elementos

- *vetor_novo*[1][2];

	0	1	2	3
vetor_novo 0				
1				
2				

Vetores, *strings* e matrizes

- Matrizes

- É possível ter mais de 2 dimensões:
- Exemplo: *char seculo[100][365][24][60][60];*
- Atenção!!! A variável do exemplo acima ocupa
 $100 * 365 * 24 * 60 * 60$ bytes = **3,1536 Gigabytes de memória**

Vetores, *strings* e matrizes

- Matrizes

- Matrizes são simplesmente uma abstração para programadores.

- Por exemplo, utilizar variáveis

char mat1[3][4];

ou

char mat2[12];

oferece os mesmos resultados.

- Muda somente a forma de acessar as posições na matriz.

Vetores, *strings* e matrizes

- Matrizes

- Exemplo: *char mat1[3][4];* e *char mat2[12];*

		0	1	2	3
mat1	0	mat2[0]	mat2[1]	mat2[2]	mat2[3]
	1	mat2[4]	mat2[5]	mat2[6]	mat2[7]
	2	mat2[8]	mat2[9]	mat2[10]	mat2[11]