

# Matrizes e Strings

---

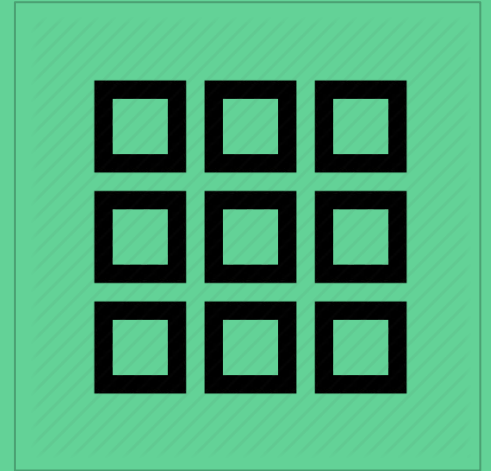
Prof. Joaquim Quinteiro Uchôa  
Profa. Juliana Galvani Gregghi  
Profa. Marluce Rodrigues Pereira  
Profa. Paula Christina Cardoso  
Prof. Renato Ramos da Silva

A grayscale photograph of a hand holding a pen, checking off a list of items on a document. The list consists of several checkboxes, some of which are already marked with an 'X'. The word 'Roteiro' is overlaid in large, bold, black letters on the image.

# Roteiro

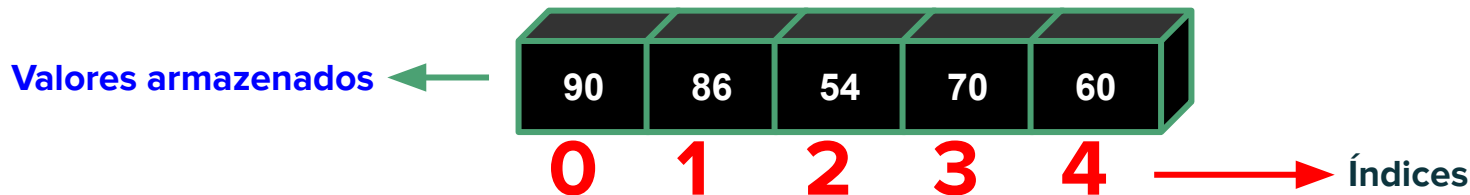
- Matrizes
- Vetores de caracteres e strings
- Vetores de strings
- Uso de `getline()`

# Matrices



# Matrizes

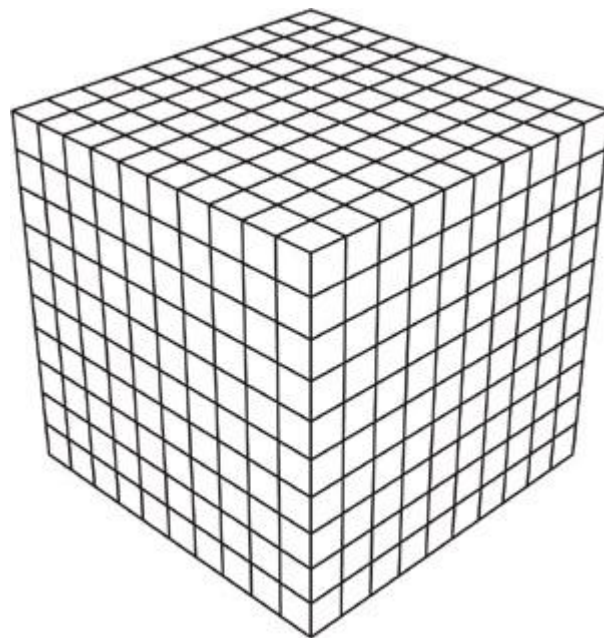
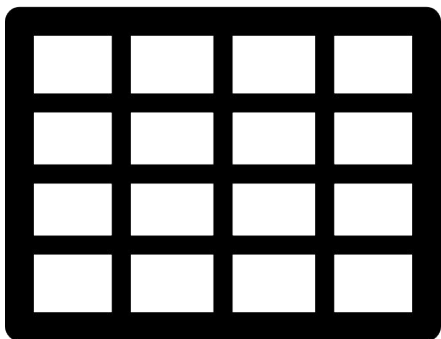
- Um arranjo é um conjunto de locais para armazenamento de elementos de um mesmo tipo de dado.
  - Também chamado de variável composta homogênea.
- Um vetor é um arranjo com uma única dimensão (unidimensional). Exemplo de um vetor:



# Matrizes

- Na prática, arranjos não precisam se restringir a uma única dimensão, podendo apresentar várias dimensões.

**Matriz (2 dimensões)**

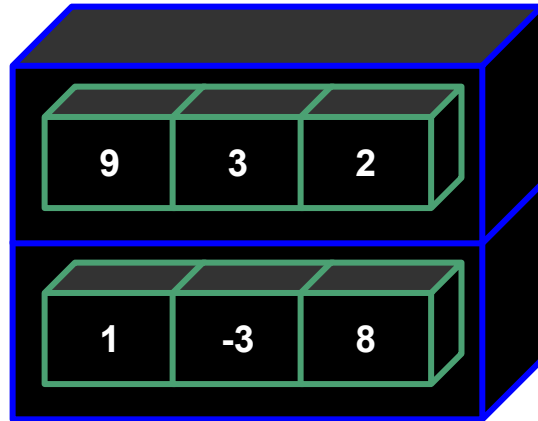


**Cubo (3 dimensões)**

# Matrizes

- **Definição:** matrizes são arranjos (variáveis compostas homogêneas) com duas ou mais dimensões.
  - Podem ser entendidas como vetores cujos elementos são outros vetores.

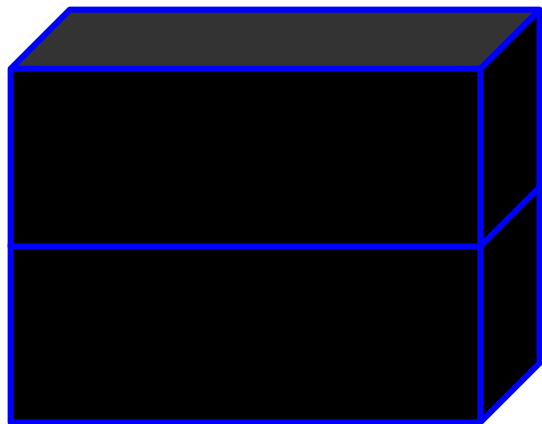
9	3	2
1	-3	8



# Matrizes

9	3	2
1	-3	8

- Neste exemplo, teríamos um vetor (mais externo) de duas posições:



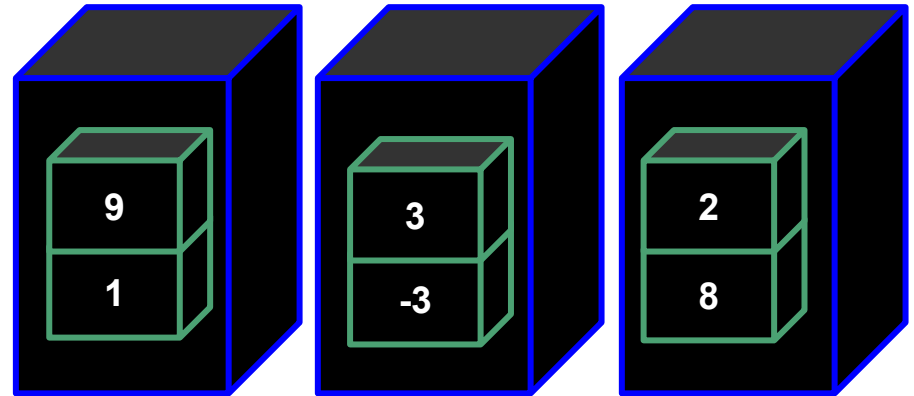
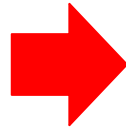
e em cada posição deste vetor (mais externo) haveria um outro vetor (mais interno) de três posições.



# Matrizes

- Poderíamos também, ao invés de enxergar os vetores por linhas, enxergá-los por coluna.

9	3	2
1	-3	8



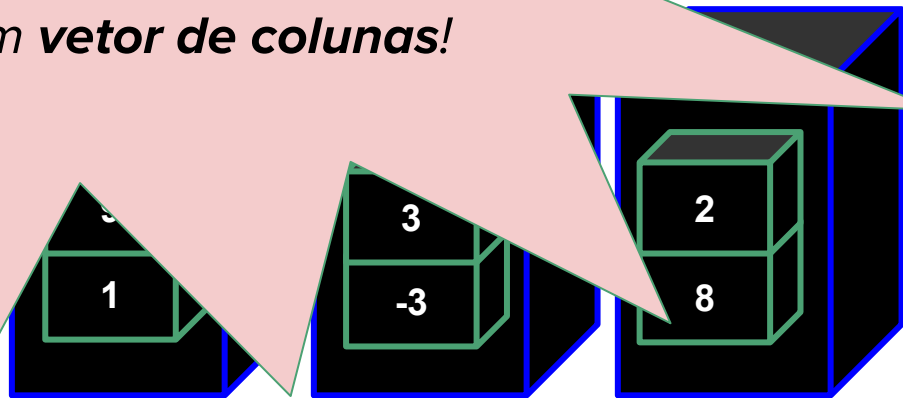


# Matrizes

- Podemos também, ao invés de organizar os vetores por linhas, organizá-los por

*Forma usual (convencional): uma matriz é um **vetor de linhas**, em que **cada linha** é um **vetor de colunas**!*

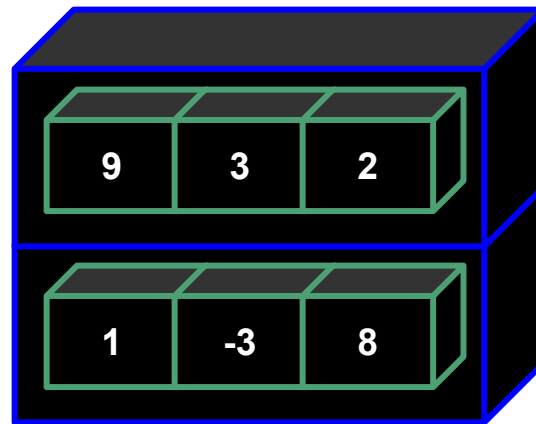
1	-3	2



# Matrizes

- Forma convencional

9	3	2
1	-3	8

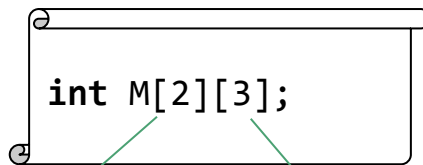


- **Sintaxe em C/C++ (declaração de uma matriz):**

- Ao declararmos uma matriz em C/C++ devemos especificar, para cada dimensão, a quantidade de elementos (posições) que aquela dimensão deve possuir.
- O tamanho de cada dimensão é especificado por um par próprio de colchetes. Exemplo de sintaxe para uma matriz com 2 dimensões:

**tipo\_dado** **identificador\_variável**[**tamanho\_dimensão\_1**][**tamanho\_dimensão\_2**];

- Exemplo:



```
int M[2][3];
```

Quantidade de linhas

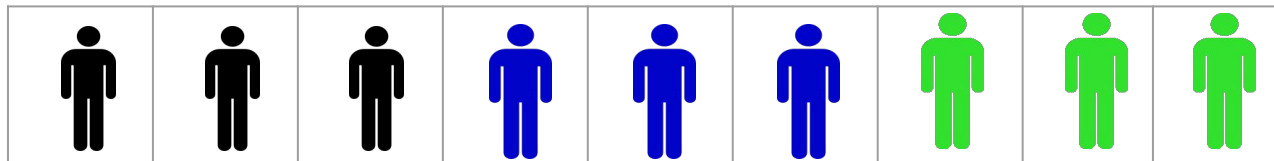
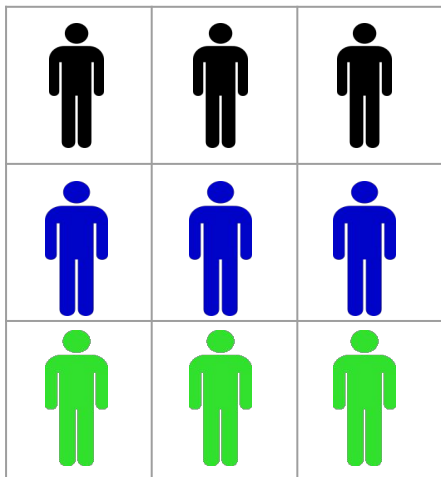
Quantidade de colunas

Neste exemplo, a matriz **M** tem capacidade de armazenar, ao todo, 6 números inteiros.

# Matrizes

## Representação em memória

- Apesar de enxergarmos uma matriz como uma estrutura bidimensional, em memória, ela é armazenada de forma linear.



# Matrizes - Exemplo

Problema: desenvolver um programa que, dada uma matriz matemática 3x3 de números reais, leia os valores a serem armazenados na matriz e escreva no dispositivo de saída padrão os valores presentes na diagonal principal da matriz, multiplicando-os por uma constante K, também fornecida pelo usuário.

Note que você poderá utilizar qualquer um dos comandos de repetição (**while**, **do..while** ou **for**) para manipular a matriz.

# Matrizes - Exemplo

Exemplo: Diagonal principal matriz 3x3

```
#include <iostream>
using namespace std;
int main(){
    int K, L = 3, C = 3; // L: qtd linhas, C: qtd colunas
    float M[3][3];
    for (int i = 0; i < L; i++)
        for (int j = 0; j < C; j++)
            cin >> M[i][j];

    cin >> K;
    for (int i = 0; i < L; i++)
        cout << M[i][i] << " * " << K << " = " << M[i][i] * K << endl;
    return 0;
}
```

# Matrizes - Exemplo

Exemplo: Diagonal principal matriz 3x3

```
#include <iostream>
using namespace std;
int main(){
    int K, L = 3, C = 3; // L: qtd linhas, C: qtd colunas
    float M[3][3];
    for (int i = 0; i < L; i++)
        for (int j = 0; j < C; j++)
            cin >> M[i][j];
    cin >> K;
    for (int i = 0; i < L; i++)
        cout << M[i][i] << " * " << K << " = " << M[i][i] * K << endl;
    return 0;
}
```

Note que como estamos trabalhando com um arranjo bidimensional, precisamos utilizar duas estruturas de repetição: uma para caminhar nos índices das linhas da matriz e outra para caminhar nos índices das colunas.

Produto da diagonal principal pela constante **K**. Obs.: sabemos que estamos na diagonal principal da matriz quando os índices das posições de linha e coluna são iguais.

# Matrizes - Exemplo

- Uso de mais dimensões
  - Lembre-se que arranjos não são limitados a apenas uma dimensão (vetores) ou 2 dimensões (matrizes).
  - Podemos desenvolver arranjos com tantas dimensões quanto necessárias. Exemplos:

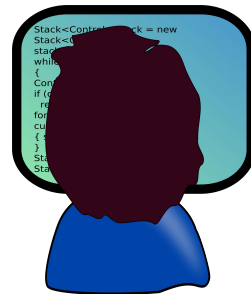
```
int cubo[3][3][3];  
int aluno[CURSO][SALA][TURMA];  
float medidas[TAM1][TAM2][TAM3][TAM4][TAM5];
```

- Para percorrer estas estruturas, usualmente utilizamos uma estrutura de repetição para cada dimensão, mas isto sempre dependerá do problema a ser resolvido.

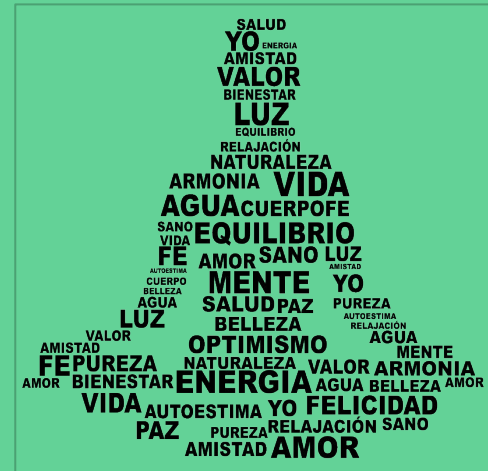


# Matrizes

- Atenção com matrizes
  - Um erro muito comum ao manipularmos uma matriz é trocarmos as variáveis que manipulam as posições de linha e coluna da matriz.
  - Por exemplo, dada uma matriz  $M[3][5]$ , erroneamente tentamos manipular a matriz na posição  $M[5][3]$ .
  - Implemente um programa que cometa este equívoco e veja qual é o erro exibido.



# Vetores de caracteres e strings



# Vetores de caracteres e strings

- Em C/C++, sequências textuais (palavras, frases, etc.) são armazenadas utilizando vetores de caracteres ou o tipo de dado **string**.
- O tipo **string**, apesar de ser considerado um tipo básico em C++, é na verdade um tipo derivado de dado que possui, internamente, um vetor de caracteres.
- Assim, independentemente de usarmos o tipo **char[]** ou o tipo **string**, podemos tratar strings em C++ exatamente como tratamos um vetor de caracteres.

# Acesso aos caracteres de uma string

As posições de uma **string** podem ser acessadas como se a **string** fosse um vetor. Exemplo:

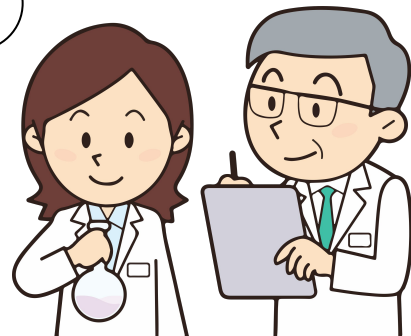
```
string frase1 = "Bom dia.";
char frase2[20] = "Boa noite.";
cout << frase1[1] << endl;
cout << frase2[6] << endl;
```



# Vetores de caracteres e strings

- Exemplo:

```
#include <iostream>
#include <string>
using namespace std;
int main(){
    string primeiroTexto = "Oa ud!";
    char segundoTexto[10] = "I,Mno!";
    int i = 0;
    while (i < 6) {
        cout << primeiroTexto[i] << segundoTexto[i];
        i++;
    }
    cout << endl;
    return 0;
}
```



Teste o programa e descubra a mensagem exibida com o **cout**.

# Vetores de caracteres e strings

- Observe que, em C, não há o tipo **string**. Em C, toda sequência de caracteres deve ser representada por meio de um vetor do tipo **char**.
- Recomenda-se usar o tipo **string**, sempre que possível, por ser mais poderoso e mais fácil de usar.
- Nem sempre é possível utilizar o tipo **string**, em algumas situações é mais adequado utilizar vetor de caracteres (em arquivos binários, por exemplo, como veremos mais à frente no curso).



# Concatenação de sequências de caracteres

Uma operação muito comum envolvendo sequências de caracteres é a concatenação, ou junção, de duas ou mais sequências.

Em C, a concatenação é feita com a função **strcat** (da biblioteca **<cstring>**). Em C++, para se concatenar duas strings basta empregarmos o operador **+** entre as strings. Exemplo:

```
string frase, nome, concatenacao;  
frase = "Bom dia, ";  
cin >> nome;  
concatenacao = frase + nome;
```



Insira este trecho de código em um programa C++ e exiba o valor da variável **concatenacao**.

# Tamanho de sequências de caracteres

Outra operação muito comum envolvendo sequências de caracteres é a obtenção do tamanho da sequência. O tamanho de uma sequência de caracteres pode ser definido como a quantidade de caracteres presentes na sequência.

Para saber o tamanho do espaço ocupado pelas sequências de caracteres, podemos utilizar a função **length()** ou **size()**, caso seja utilizado o tipo **string** ou a função **strlen()** no caso de vetores do tipo **char**.



# Tamanho de sequências de caracteres

## Sintaxe para strings:

```
identificador_variável.length()  
ou  
identificador_variável.size()
```

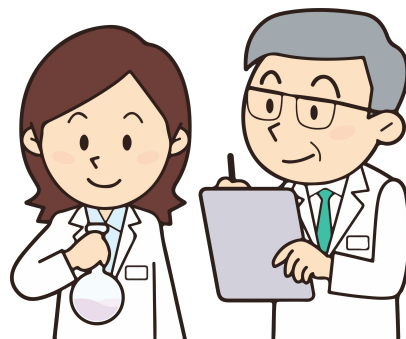
## Sintaxe para vetores de char:

```
#include <cstring> //no início do código (antes da função main())  
  
strlen(identificador_variável)
```

# Tamanho de sequências de caracteres

Exemplo:

```
#include <iostream>
#include <cstring> // apenas para strlen()
using namespace std;
int main(){
    int tam1, tam2;
    string frase1 = "Bom dia.";
    char frase2[30] = "Boa noite.";
    tam1 = frase1.length(); // ou frase1.size()
    tam2 = strlen(frase2);
    cout << tam1 << " " << tam2 << endl;
    return 0;
}
```



# Codificação de caracteres

Caracteres são representados utilizando algum tipo de codificação. Ou seja, há uma definição que relaciona um determinado código (valor numérico) com caractere em específico, seja ele uma letra, um dígito ou um símbolo.

O padrão de codificação básico das linguagens de programação é conhecido como ASCII.

Outro padrão, mais recente e completo, é chamado Unicode.

# ASCII TABLE

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]

# Strings × Vetores de Caracteres

- O último caractere de strings em C/C++ é o caractere nulo, representado pela sequência de escape `\0`, que tem o valor **0** na tabela ASCII.
- O caractere nulo indica o fim de strings. Por exemplo:

```
char letras1[4] = {'i', 'a', 'l', 'g'};  
char letras2[5] = {'i', 'a', 'l', 'g', '\0'};
```

Apenas a variável **letras2** é considerada uma string, pois termina em `\0`. A variável **letras1** é considerada simplesmente como um vetor de caracteres.

# Um exemplo usando vetor de char

```
//Substituir cada letra da palavra pela sua antecessora no alfabeto
// abacaxi -> bcbdbbyj
// OBS: a letra 'z' deve ser substituída pela letra 'a'
#include <iostream>
using namespace std;

int main(){
    char palavra[12]; //palavras com no máximo 12 caracteres

    cin>>palavra; //lendo uma entrada e armazenando-a no vetor

    int indicePalavra = 0;
    while (indicePalavra < 12 and palavra[indicePalavra] != '\0'){
        if (palavra[indicePalavra] == 'z')
            palavra[indicePalavra] = 'a';
        else
            palavra[indicePalavra] = palavra[indicePalavra] + 1;
        indicePalavra++;
    }
    cout<<palavra<<endl;
    return 0;
}
```

# Um exemplo usando vetor de char

```
//Substituir cada letra da palavra pela sua antecessora no alfabeto
// abacaxi -> bcbdbj
// OBS: a letra 'z' deve ser substituída pela letra 'a'
#include <iostream>
using namespace std;

int main(){
    char palavra[12]; //palavras com no máximo 12 caracteres

    cin>>palavra; //lendo uma entrada e armazenando no vetor

    int indicePalavra = 0;
    while (indicePalavra < 12 and palavra[indicePalavra] != '\0'){
        if (palavra[indicePalavra] == 'z')
            palavra[indicePalavra] = 'a';
        else
            palavra[indicePalavra] = palavra[indicePalavra] + 1;
        indicePalavra++;
    }
    cout<<palavra<<endl;
    return 0;
}
```

Teste para verificar se a palavra já não terminou.

***O que aconteceria se esse teste não fosse feito?***

# Caractere não é string

Uma variável **string**, terminada em `\0`, é diferente de uma variável do tipo caractere. Por exemplo:



“z” não é o mesmo que ‘z’



# Biblioteca <string>

C++ oferece uma série de funções para trabalhar com strings:

- Busca: função **find()**.
- Remoção de um caractere em uma posição específica da sequência: função **erase()**.
- Substituição: função **replace()**.
- Entre outras...

# Biblioteca <string>

Para utilizar tais métodos, o programador deve inserir a biblioteca string (**#include <string>**) em seu código.

Para mais detalhes, verifique:

- <http://www.cplusplus.com/reference/string/string/>
- <http://www.cplusplus.com/forum/beginner/4614/>



# Vetores de Strings



# Particularidades de vetores de strings - i

Em muitos problemas é necessário percorrer um **vetor de strings** para resolver um dado problema.

Suponha, por exemplo, que você queira verificar se um dado arquivo texto possui uma certa palavra. Além disso, para outras finalidades, essas palavras devem ser lidas em um vetor com capacidade para no máximo 100 palavras.

# Particularidades de vetores de strings - i

Esse problema é resolvido com relativa facilidade lendo palavra por palavra do vetor, usando uma estrutura de repetição (while ou for), e comparando com a palavra procurada.

Usando a classe string, basta usar o comparador “==”.

Se utilizado vetor de char, basta usar a função `strcmp( )`, da biblioteca `<cstring>`.

# Exemplo usando vetor de strings

```
int main(){
    ifstream arqEntrada ("entrada.txt");
    string procurada, palavras[100]; //vetor de strings
    int totalPalavras = 0, indicePalavra = 0;
    bool encontrada = false; //sentinela para interromper a
                             //se a palavra for encontrada

    cin>>procurada;

    if (arqEntrada){
        while (indicePalavra < 100 and
               arqEntrada>>palavras[indicePalavra]){
            indicePalavra++;
        }
    }
    totalPalavras = indicePalavra;
    indicePalavra = 0;
    while (indicePalavra < totalPalavras and !encontrada){
        if (palavras[indicePalavra] == procurada){
            cout<<"Palavra encontrada"<<endl;
            encontrada = true;
        }
        indicePalavra++;
    }
    if (!encontrada)
        cout<<"Palavra não encontrada";
}
```

# Mas e se eu quiser verificar caracteres? - i

Suponha agora que eu queira verificar todas as palavras em um arquivo texto que possuam pelo menos três vogais.

Suponha, da mesma forma que no exercício anterior, que eu precise ler o conteúdo do arquivo em um vetor com capacidade para 100 palavras.

E agora? Eu não vou comparar palavras entre si, eu preciso **verificar cada palavra**, usando o fato que **strings são vetores**.

## Mas e se eu quiser verificar caracteres? - ii

Nesse caso, nós vamos ter um vetor de palavras, um vetor de strings. Na prática, um **vetor de vetor de caracteres**. Então, precisamos resolver esse problema de maneira similar ao que fazemos com matrizes.

### Como assim?

Precisamos de **duas estruturas de repetição, uma dentro da outra**. Uma para percorrer o vetor de palavras... E outra para percorrer cada palavra como um vetor de carácter.



# Exemplo (1/2)

```
#include <iostream>
#include <string>
#include <fstream>
using namespace std;

int main(){
    ifstream arqEntrada ("entrada.txt");
    string maisAs, palavras[100]; //vetor de strings
    int totalVogais, indicePalavra, indiceLetra, tamanhoPalavra;
    indicePalavra = 0;
    if (arqEntrada){
        while (indicePalavra < 100 and
            arqEntrada >> palavras[indicePalavra]){
            indicePalavra++;
        }
    }

    int totalPalavras = indicePalavra;
    indicePalavra = 0;
    indiceLetra = 0;
```

## Exemplo (2/2)

```
while (indicePalavra < totalPalavras){
    tamanhoPalavra = palavras[indicePalavra].size();
    totalVogais = 0;
    while (indiceLetra < tamanhoPalavra){
        if (palavras[indicePalavra][indiceLetra] == 'a' or
            palavras[indicePalavra][indiceLetra] == 'e' or
            palavras[indicePalavra][indiceLetra] == 'i' or
            palavras[indicePalavra][indiceLetra] == 'o' or
            palavras[indicePalavra][indiceLetra] == 'u')
            totalVogais++;
        indiceLetra++;
    }

    if (totalVogais >= 3)
        cout<<palavras[indicePalavra]<<endl;

    indicePalavra++;
}
return 0;
}
```

## Exemplo (2/2)

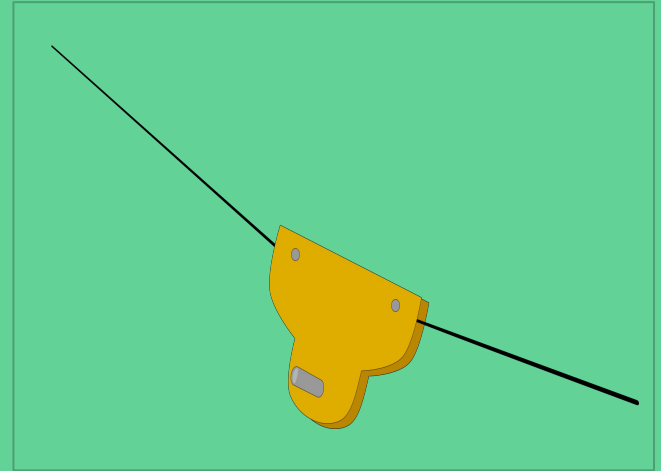
```
while (indicePalavra < totalPalavras){
    tamanhoPalavra = palavras[indicePalavra].size();
    totalVogais = 0;
    while (indiceLetra < tamanhoPalavra){
        if (palavras[indicePalavra][indiceLetra] == 'a' or
            palavras[indicePalavra][indiceLetra] == 'e' or
            palavras[indicePalavra][indiceLetra] == 'i' or
            palavras[indicePalavra][indiceLetra] == 'o' or
            palavras[indicePalavra][indiceLetra] == 'u')
            totalVogais++;
        indiceLetra++;
    }

    if (totalVogais >= 3)
        cout<<palavras[indicePalavra]<<endl;

    indicePalavra++;
}
return 0;
}
```

Usamos dois  
índices: um para a  
palavra, outro  
para a letra... de  
maneira similar a  
uma matriz...

# Uso de getline()



# Leitura de strings com espaço

- Para receber uma **string** do dispositivo de entrada padrão, podemos utilizar em C++ o comando **cin**.
- No entanto, será considerado como parte da **string** somente os caracteres até um espaço em branco qualquer (seja um espaço simples ou uma quebra de linha).
- Caso seja necessário fazer a leitura de uma **string** que inclua espaços simples em sua sequência de caracteres (como, por exemplo, um nome composto ou uma frase completa), podemos utilizar para isso a função **getline()**.

# Leitura de strings com espaço

Sintaxe em C++ do getline para strings:

```
getline(cin, identificador_variável);
```

em que **identificador\_variável** deve ser o nome de uma variável do tipo **string**.  
Exemplo:

```
string frase1;  
getline(cin, frase1);
```

# Leitura de strings com espaço

## Sintaxe em C++ do getline para vetores de char:

```
cin.getline(identificador_variável, quantidade_caracteres);
```

em que **identificador\_variável** deve ser o nome de uma variável do tipo vetor de **char** e **quantidade\_caracteres** é a quantidade de caracteres que será considerada no processo de leitura, sendo que o último deles automaticamente será o caractere nulo. Exemplo:

```
char frase2[20];  
cin.getline(frase2,20);
```

# Cuidados ao usar `getline()`

Imagine que você deva fazer um programa que leia do teclado o endereço de e-mail de um usuário, seu nome e sua idade, exibindo posteriormente estas informações no monitor. Considere que ao fornecer o endereço de e-mail, o usuário nunca digitará um espaço em branco no meio do endereço, enquanto que nomes compostos são permitidos.

Você irá se deparar com resultados inesperados ao resolver o problema, caso use o `getline()`, sem tomar os devidos cuidados.



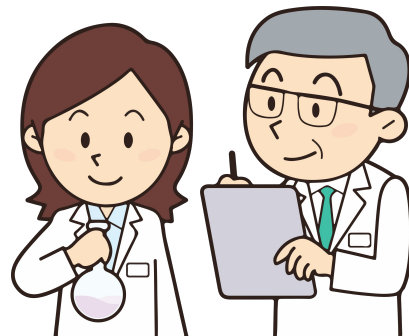


# Cuidados ao usar getline()



Exemplo com erro

```
#include <iostream>
using namespace std;
int main() {
    string nome, email;
    int idade;
    cin>>email;
    getline(cin,nome);
    cin >> idade;
    cout << "Nome: " << nome << endl;
    cout << "Email: " << email << endl;
    cout << "Idade: " << idade << endl;
    return 0;
}
```



Embora possa parecer que este programa resolva o problema apresentado, ele na verdade possui um grave erro de leitura dos dados. Teste você mesmo e veja o que acontece.

# Cuidados ao usar `getline()`

Exemplo de execução do programa anterior com o erro durante a leitura:

```
lovelace@pascal.com  
Ada Lovelace  
Nome:  
Email: lovelace@pascal.com  
Idade: 0
```

Note que ao executarmos o programa, o usuário não consegue fornecer a informação da **idade** do usuário. Além disso, não é exibida nenhuma informação sobre o **nome** fornecido no dispositivo de entrada.

**Por que isso aconteceu?**

# Cuidados ao usar `getline()`

Para compreender o que aconteceu, você deve se recordar de que: ao executarmos o comando **`cin`** para uma **`string`**, será considerado como parte da **`string`** somente os caracteres até um espaço em branco qualquer (seja um espaço simples, uma tabulação ou uma quebra de linha).

Sendo assim, ao executarmos o **`cin`** da linha 6 (**`cin >> email;`**), carregamos na variável `email` apenas os caracteres do e-mail fornecido, desconsiderando o caractere de quebra de linha ao terminar de digitar o e-mail. **A questão agora é:** o que acontece com este caractere de quebra de linha do final do e-mail?



# Cuidados ao usar `getline()`

Como o próximo comando de leitura que realizamos é o `getline` da linha 7 (`getline(cin,nome);`), a quebra de linha do final do e-mail fornecido é carregada na variável **nome**.

Por isso, a exibição desta variável na linha 9 (`cout << "Nome: " << nome << endl;`) não exibe “nada”. Na verdade, pela forma como o programa foi escrito, estamos exibindo para o **nome** apenas uma quebra de linha.

(...)

# Cuidados ao usar `getline()`

(...)

Por fim, é executada a leitura da linha 8 (`cin >> idade;`), que no exemplo de execução apresentado, seria o equivalente a tentar atribuir o valor textual do nome digitado na variável **idade**, que é do tipo inteiro.

Por isso, a exibição da variável **idade** na linha 11 (`cout << "Idade: " << idade << endl;`), apresentou o valor 0, pois houve uma tentativa de atribuir uma **string** a uma variável inteira. Houve erro na leitura.

# Cuidados ao usar `getline()`

Qual a solução para este problema?

- Todo o problema foi ocasionado devido a leitura da quebra de linha pelo `getline` na linha 7 (`getline(cin,nome);`) na variável **nome**.
- Para resolvermos este problema, basta indicarmos que a leitura do `getline` desta variável deve ignorar este primeiro caractere de quebra de linha. Em C++, podemos fazer isso por meio do comando:

```
cin.ignore();
```

# Cuidados ao usar getline()

Exemplo corrigido

```
#include <iostream>
using namespace std;
int main(){
    string nome, email;
    int idade;
    cin>>email;
    cin.ignore();
    getline(cin,nome);
    cin >> idade;
    cout << "Nome: " << nome << endl;
    cout << "Email: " << email << endl;
    cout << "Idade: " << idade << endl;
    return 0;
}
```

A inserção do comando ignore deve ser feita aqui.



Teste a nova versão e veja se agora funciona normalmente.

# Cuidados ao usar `getline()`

Exemplo de execução do programa anterior corrigido com o `ignore()`:

```
lovelace@pascal.com  
Ada Lovelace  
30  
Nome: Ada Lovelace  
Email: lovelace@pascal.com  
Idade: 30
```

Observe que se o mesmo programa apresentasse apenas comandos de leitura diretos com **`cin`**, ou seja, sem nenhum uso de `getline`, então não precisaríamos utilizar o comando `ignore()` em nenhum momento.





# Sobre o Material



# Sobre este material

Material produzido coletivamente, principalmente pelos seguintes professores do DCC/UFLA:

- Janderson Rodrigo de Oliveira
- Joaquim Quinteiro Uchôa
- Juliana Galvani Greggi
- Renato Ramos da Silva

Inclui contribuições de outros professores do setor de Fundamentos de Programação do DCC/UFLA.

Esta obra está licenciado com uma Licença [Creative Commons Atribuição 4.0 Internacional](https://creativecommons.org/licenses/by/4.0/).