



Strings e Vectors em C++

Professores:
Álvaro Coelho, Edgar Alexander, Esbel
Valero e Hélder Almeida

INSTITUIÇÃO EXECUTORA



COORDENADORA



APOIO



Strings

- Conceito

- Linha de caracteres (normalmente representadas pela tabela ASCII)
- Com um terminador
 - Caracter '\0' → 0 na Tabela ASCII



Tabela ASCII

Characters and Symbols from the ASCII Table

Capital Letters

Symbol	Decimal
A	65
B	66
C	67
D	68
E	69
F	70
G	71
H	72
I	73
J	74
K	75
L	76
M	77
N	78
O	79
P	80
Q	81
R	82
S	83
T	84
U	85
V	86
W	87
X	88
Y	89
Z	90

Smaller case letters

Symbol	Decimal
a	97
b	98
c	99
d	100
e	101
f	102
g	103
h	104
i	105
j	106
k	107
l	108
m	109
n	110
o	111
p	112
q	113
r	114
s	115
t	116
u	117
v	118
w	119
x	120
y	121
z	122

Non-alpha symbols 1

Symbol	Decimal
<Space>	32
!	33
-	34
#	35
\$	36
%	37
&	38
.	39
{	40
)	41
*	42
+	43
,	44
-	45
_	46
/	47
:	58
:	59
<	60
=	61
>	62
?	63

Non-alpha symbols 2

Symbol	Decimal
@	64
[91
\	92
]	93
^	94
-	95
{	123
	124
)	125
~	126

Fonte: <https://www.tes.com/teaching-resource/characters-and-symbols-ascii-table-12255608>

Tabela ASCII

ASCII control characters		ASCII printable characters				Extended ASCII characters			
00	NULL (Null character)	32	space	64	@	96	`	128	ç
01	SOH (Start of Header)	33	!	65	A	97	a	129	ú
02	STX (Start of Text)	34	"	66	B	98	b	130	é
03	ETX (End of Text)	35	#	67	C	99	c	131	â
04	EOT (End of Trans.)	36	\$	68	D	100	d	132	ã
05	ENQ (Enquiry)	37	%	69	E	101	e	133	à
06	ACK (Acknowledgement)	38	&	70	F	102	f	134	à
07	BEL (Bell)	39	'	71	G	103	g	135	ç
08	BS (Backspace)	40	(72	H	104	h	136	ê
09	HT (Horizontal Tab)	41)	73	I	105	i	137	ë
10	LF (Line feed)	42	*	74	J	106	j	138	è
11	VT (Vertical Tab)	43	+	75	K	107	k	139	í
12	FF (Form feed)	44	,	76	L	108	l	140	î
13	CR (Carriage return)	45	-	77	M	109	m	141	ï
14	SO (Shift Out)	46	.	78	N	110	n	142	Ã
15	SI (Shift In)	47	/	79	O	111	o	143	Ã
16	DLE (Data link escape)	48	0	80	P	112	p	144	É
17	DC1 (Device control 1)	49	1	81	Q	113	q	145	æ
18	DC2 (Device control 2)	50	2	82	R	114	r	146	Æ
19	DC3 (Device control 3)	51	3	83	S	115	s	147	ô
20	DC4 (Device control 4)	52	4	84	T	116	t	148	ö
21	NAK (Negative acknowl.)	53	5	85	U	117	u	149	ò
22	SYN (Synchronous idle)	54	6	86	V	118	v	150	û
23	ETB (End of trans. block)	55	7	87	W	119	w	151	ù
24	CAN (Cancel)	56	8	88	X	120	x	152	ÿ
25	EM (End of medium)	57	9	89	Y	121	y	153	Ö
26	SUB (Substitute)	58	:	90	Z	122	z	154	Ü
27	ESC (Escape)	59	;	91	[123	{	155	ø
28	FS (File separator)	60	<	92	\	124		156	£
29	GS (Group separator)	61	=	93]	125	}	157	Ø
30	RS (Record separator)	62	>	94	^	126	~	158	×
31	US (Unit separator)	63	?	95	-			159	f
127	DEL (Delete)							223	nbsp

Fonte: <https://www.yoair.com/blog/an-overview-of-the-ascii-table-american-system-code-for-information-interchange/> 4



Exemplos de Strings

Pos	0	1	2	3	4	5	6	7	8	9	10
Ch	R	e	s	i	d	e	n	c	i	a	\0

Pos	0	1	2	3	4	5	6	7	8	9	10
Ch	H	o	j	e		t	e	m	!	\0	

Pos	0	1	2	3	4	5	6	7	8	9	10
Ch	V	i	m	,	v	i		e		\0	

Strings em C

- Strings são implementados como Arrays
 - De char
- Ex. `char nome[50]` ← array com 50 itens do tipo char
- Podemos percorrer uma string
 - For com um int pra indexar

```
#include <iostream>

int main()
{
    char nome[10] = "Olá Mundo";
    for (int i=0; i<10; i++)
        std::cout << nome[i] << "\n";
}
```

Strings em C

- Observe (e execute) o programa ao lado.
- Qual o valor que ele mostra para X?
- Porque?

```
#include <iostream>

int main()
{
    char nome[10] = "Olá Mundo";
    char x;
    nome[10] = 'G';
    std::cout << x;
}
```



Residência
em Software

Classe string

Pos	0	1	2	3	4	5	6	7	8	9	10	11
Ch	R	e	s	i	d	e	n	c	i	a	\0	...

swap()

copy()

...

length()

at()

...

Operador +

Operador =

...

...

...

...

- Para usar é necessário solicitar um “serviço”
 - O Serviço previne operações ilegais
 - Menos acesso à estrutura interna → menos chance de infringir a consistência
- Encapsulamento

Classe string (C++)

- Definida na biblioteca string
 - `#include <string>`
- Declarada como uma variável qualquer (tipo string)
 - Pode ser inicializada no construtor
 - `string nome("Residencia");`



Residência
em Códigos

```
#include <iostream>
#include <string>      // Necessário para usar strings

int main()
{
    std::string nome1("Fulano");      // Inicializa nome1
    std::string nome2("Beltrano");    // Inicializa nome2
    std::string nome3, nome4;        // Não inicializa nome3 nem nome4
    std::cout << "Os dois primeiros nomes são "
    |   << nome1 << " e " << nome2 << std::endl;
    nome3 = "Ciclano";            // Inicialização posterior de nome3
    nome4 = nome3;
    std::cout << "O terceiro e quarto nomes são "
    |   << nome3 << " e " << nome4 << std::endl;
}
```

Observe os exemplos

```
#include <iostream>
#include <string>      // Necessário para usar strings

using namespace std;

int main()
{
    string nome1("Fulano");      // Inicializa nome1
    string nome2("Beltrano");    // Inicializa nome2
    string nome3, nome4;        // Não inicializa nome3 nem nome4
    cout << "Os dois primeiros nomes são "
    |   << nome1 << " e " << nome2 << endl;
    nome3 = "Ciclano";          // Inicialização posterior de nome3
    nome4 = nome3;
    cout << "O terceiro e quarto nomes são "
    |   << nome3 << " e " << nome4 << endl;
}
```

Observe o uso de endl, cout, cin e string em cada exemplo

Namespace

- Muitas vezes há o risco de nomes de recursos coincidirem
 - Recurso = dado (variável) ou função
- Uma solução é usar o caminho completo pra evitar ambiguidade
 - std::string
- Outra é definir um espaço de nomes para o escopo
 - using namespace std

Manipulando strings

- Entrada de dados
 - getline() ou cin >>
- Saída de dados
 - cout <<
- Observe o programa ao lado
 - Trocar getline() por cin >> dá em que?

```
#include <iostream>
#include <string>           // Necessário para usar strings

using namespace std;

int main()
{
    string nome;
    cout << "Digite seu nome: ";
    getline(cin, nome);
    cout << endl << "Seu nome eh " << nome;
}
```

Atribuição de Strings

- nome1 = nome2;
- Em C isto não é possível
 - São endereços de memória
- Classe string implementa operadores
 - = ← operador de atribuição
 - Aceita constantes e variáveis

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

int main()
{
    string nome1("Bart");
    string nome2("Homer");
    string nome3, nome4;

    cout << "Os dois primeiros nomes sao "
        << nome1 << " e " << nome2 << endl;
    nome3 = "Marge";
    nome4 = nome3;
    cout << "O terceiro e quarto nomes sao "
        << nome3 << " e " << nome4 << endl;
}
```

Acessando caracteres

- Função `at()`
 - Recupera o caracter na posição indicada
- Observe o programa ao lado
- Corremos o risco de “invadir” outra variável (acessando uma posição “além” da string?)
 - Porque?

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

int main()
{
    string nome1("Residencia");
    for (int i=0;i<nome1.length(); i++)
        cout << nome1.at(i) << endl;
}
```

Operações e Contextos

- Muitas vezes queremos o conforto de usar uma mesma operação em contextos diferentes.
 - Por exemplo: se estamos no contexto de números inteiros, $a+b$ é uma operação de soma direta.
 - No contexto de números complexos, $a+b$ é uma operação diferente
 - No contexto de strings, $a+b$ significará a concatenação entre elas

Concatenando strings

- Operador + (a+b)
 - Admite-se também a+=b
 - Em C não é possível pois são endereços
- Observe o programa ao lado
- E se fosse para pedir a digitação do primeiro e o último nome de alguém e depois escrever o nome completo?
 - Observe que quem digita talvez não se preocupe com espaços!

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

int main()
{
    string nome1("Residencia");
    string nome2(" de Software");
    string nome3 = nome1 + nome2;
    cout << nome3;
}
```

Mais funções de strings

- **String.swap()**
 - Para trocar o conteúdo de duas strings
 - Em C seria necessário fazer uma cópia
- Observe o programa ao lado

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

int main()
{
    string nome1 = "Joaquim";
    string nome2 = "Nabuco";
    cout << nome1 << endl;
    cout << nome2 << endl;
    nome1.swap(nome2);
    cout << nome1 << endl;
    cout << nome2 << endl;
}
```



Residência
em Software

Voltaremos a usar strings no exercício de revisão

Vectors

- Conceito
 - Um objeto para registrar uma coleção de objetos
 - Qualquer objeto pode ser colecionável
 - Impõe uma indexação (e iteração)
 - Há outros tipos de coleções (list, queue, ...)
- Representação
 - É o encapsulamento de um array

Arrays e Vectors

Pos	0	1	2	3	4	5	6	7	8	9	10	11
Int	0	1	1	2	3	5	8	13	21	34	55	...

- Observe um array na memória
- Observe o programa ao lado
 - O que é mostrado? Porque? Qual a fragilidade?

```
#include <iostream>

main() {
    int vetor[11] = {0,1,1,2,3,
                    ...           5,8,13,21,34,55};
    int k = 0;
    vetor[11] = -1;
    std::cout << k;
}
```

Mais “problemas” com arrays

- Cada variável array é, na verdade um endereço
 - int a[10], b[20]
 - a e b são números de 64 (ou 32, ou 128) bits: uma posição da memória (ponteiro)
 - Quando acessamos um elemento (por exemplo cout << a[3])
 - O compilador gera um código que desloca 3 posições a partir do endereço de a
 - Por isso que “invade” outra variável
- O que acontece se tentamos fazer a=b?

Mais problemas com arrays

- Adicionar um elemento
- Veja o exemplo ao lado*.
 - Para acrescentar um elemento foi necessário
 - Redimensionar o array
 - Deslocar os elementos além da posição 2
 - Acrescentar o elemento
 - *Poderia ser feito com ponteiros e alocação dinâmica

```
#include <iostream>

int main () {
    int vetor[11] = {0,1,2,3,
                    5,8,13,21,34,55};
    //queremos inserir o numer 1 na posição 2
    vetor[11];
    for (int i=10;i>=2;i--) //deslocando os posteriores a 2
        vetor[i+1] = vetor[i];
    vetor[2] = 1; //inserindo o 1 na posição 2
    for (int i=0;i<11;i++) //listar o vetor modificado
        std::cout << vetor[i] << "\n";
}
```



Classe vector

Exemplo de Objeto Vector

Pos	0	1	2	3	4	5	6	7	8	9	10	11
Int	0	1	1	2	3	5	8	13	21	34	55	...

size()	begin()	at()	...
empty()	end()	Operador =	...
...

- Encapsulamento
 - Não se tem acesso aos elementos diretamente
 - Apenas através de funções (métodos)
 - As funções “resolvem” os problemas potenciais
 - Atribuição da responsabilidade de gerenciar o vetor à classe vector

Classe vector

- Declaração
 - `vector<tipo> nome;`
 - Ex. `vector<float> notas;`
- Declaração com inicialização
 - No construtor (função especial usada para definir valor inicial de qualquer objeto)

```
#include <iostream>
#include <vector>

int main () {
    std::vector<int> vec {0, 1,
                         1, 2, 3, 5, 8, 13, 21};
```

Inserindo dados num vetor

- No final (operação mais comum)
 - vetor.push_back()
- Observe no programa ao lado
 - O que aparece?
 - Método size()

```
#include <iostream>
#include <vector>

using namespace std;

int main () {
    vector<int> vec {0, 1,
                    1, 2, 3, 5, 8, 13, 21};
    cout << vec.size() << endl;
    vec.push_back(34);
    cout << vec.size() << endl;
}
```

Algumas funções

- `vetor.at()`
- Localiza um elemento numa dada posição
 - Limitado ao tamanho do vetor (`size()`)
- Observe o programa ao lado
 - O que é exibido?
 - O que aconteceria se o comando fosse `vet.at(10)`?

```
#include <iostream>
#include <vector>

using namespace std;

int main () {
    vector<int> vec {0, 1,
                    1, 2, 3, 5, 8, 13, 21};
    cout << vec.at(3) << endl;
}
```

Iterators

- Objetos usados para percorrer coleções
- Observe o programa ao lado
 - Estamos *iterando* sobre o array!

```
#include <iostream>

using namespace std;

main() {
    int vetor[10] = {0,1,2,3,
                    5,8,13,21,34,55};
    for (int i=0;i<11;i++)
        cout << vetor[i] << endl;
}
```

Criando um objeto iterator

- Um iterator é um ponteiro
 - Indica a *posição* onde um item da coleção está
 - Observe a sintaxe *
 - *it significa “conteúdo da posição indicada por it”
- Observe que usamos funções begin() e end()
 - Limites do objeto vector

```
#include <iostream>
#include <vector>

using namespace std;

int main () {
    vector<int> vec {0, 1,
                     1, 2, 3, 5, 8, 13, 21};
    for (auto it = vec.begin();
         it != vec.end(); it++) {
        cout << *it << " ";
    }
}
```

- `advance(it,n)` serve para fazer um iterator avançar n posições pela coleção
- Observe o exemplo abaixo
- Observe a declaração do iterator “não automático”
 - Membro da coleção `vector<int>`

Advance()

```
#include <iostream>
#include <vector>

using namespace std;

int main () {
    vector<int> vec {0, 1,
                    1, 2, 3, 5, 8, 13, 21};

    vector<int>::iterator it;
    it = vec.begin();
    cout << *it << endl;
    advance(it,1);
    cout << *it << endl;
    advance(it,4);
    cout << *it << endl;
}
```



- Muitas vezes queremos saber quantos *hops* de distância há entre dois iteradores
 - `distance(i1,i2)`

distance()

```
#include <iostream>
#include <vector>

using namespace std;

int main () {
    vector<int> vec {0, 1,
                    1, 2, 3, 5, 8, 13, 21};

    vector<int>::iterator it;
    it = vec.begin();
    advance(it,5);
    cout << "Distancia de " <<
          distance(vec.begin(), it) <<
          " posicoes" << endl;
}
```

Manipulando dados em vectors

- Até aqui vimos meras operações de localização e consulta de dados em vectors
- Manipulação
 - Inserir
 - Já vimos `push_back()`
 - E para inserir em posições intermediárias?
 - Excluir
 - Alterar

Inserir: insert()

- Todo vector pode ter invocada a função `insert(it, v)`
 - Insere `v` na posição indicada pelo iterator `it`
- `insert()` cuida de ajustar as posições de todos os demais itens

```
#include <iostream>
#include <vector>

using namespace std;

int main () {
    vector<int> vec {0, 1,
                     1, 2, 3, 5, 8, 13, 21};
    for (auto x = vec.begin();
         x != vec.end(); x++)
        cout << *x << endl;
    vector<int>::iterator it;
    it = vec.begin();
    advance(it,5);
    vec.insert(it,-7);
    cout << "-----" << endl;
    for (auto x = vec.begin();
         x != vec.end(); x++)
        cout << *x << endl;
}
```

Apagar: erase()

- Todo vector pode ter invocada uma função `erase(it)`
 - Apaga o item na posição indicada pelo iterador `it`
- `erase()` cuida de ajustar a posição de todos os demais itens

```
#include <iostream>
#include <vector>

using namespace std;

int main () {
    vector<int> vec {0, 1,
                    1, 2, 3, 5, 8, 13, 21};
    for (auto x = vec.begin();
         x != vec.end(); x++)
        cout << *x << endl;
    vector<int>::iterator it;
    it = vec.begin();
    advance(it,5);
    vec.erase(it);
    cout << "-----" << endl;
    for (auto x = vec.begin();
         x != vec.end(); x++)
        cout << *x << endl;
}
```

Alterar

- Um iterator é um ponteiro para um item
- Alterar: mudar o *conteúdo* da posição indicada pelo iterator
 - `*it = -1`
 - O dado da posição indicada por `it` se torna -1

```
#include <iostream>
#include <vector>

using namespace std;

int main () {
    vector<int> vec {0, 1,
                    1, 2, 3, 5, 8, 13, 21};
    for (auto x = vec.begin();  
         x != vec.end(); x++)
        cout << *x << endl;
    vector<int>::iterator it;
    it = vec.begin();
    advance(it,5);
    *it = -1;
    cout << "-----" << endl;
    for (auto x = vec.begin();  
         x != vec.end(); x++)
        cout << *x << endl;
}
```



Alterar – Exemplo com string

```
#include <iostream>
#include <vector>
#include <string>

using namespace std;

int main () {
    vector<string> vec;
    vec.push_back("Zangado");
    vec.push_back("Atchim");
    vec.push_back("Mestre");
    vec.push_back("Soneca");
    vec.push_back("Dunga");
    vec.push_back("Feliz");
    vec.push_back("Dengoso");
    for (auto x = vec.begin();
         x != vec.end(); x++)
        cout << *x << endl;
    vector<string>::iterator it;
    it = vec.begin();
    advance(it,5);
    *it = "Branca";
    cout << "-----" << endl;
    for (auto x = vec.begin();
         x != vec.end(); x++)
        cout << *x << endl;
}
```

Tamanho de um vector: size()

- Todo vector possui uma função `size()`
 - Retorna um inteiro indicando quantos itens há no vector

```
#include <iostream>
#include <vector>
#include <string>

using namespace std;

int main () {
    vector<string> vec;
    cout << vec.size() << endl;
    vec.push_back("Zangado");
    vec.push_back("Atchim");
    vec.push_back("Mestre");
    vec.push_back("Soneca");
    vec.push_back("Dunga");
    vec.push_back("Feliz");
    vec.push_back("Dengoso");
    cout << vec.size() << endl;
}
```

Percorrer um vector ao contrário

- Muitas vezes queremos iterar um vector no sentido inverso
 - Podemos usar begin() e end() como limites
 - Repetindo o fato de que “começa” em end() -1 e “termina” em begin()-1
 - Iterando negativamente (--)

```
#include <iostream>
#include <vector>
#include <string>

using namespace std;

int main () {
    vector<int> vec {0, 1,
                    1, 2, 3, 5, 8, 13, 21};
    for (auto x = vec.begin();
         x != vec.end(); x++)
        cout << *x << endl;
    cout << "-----" << endl;
    for (auto x = vec.end()-1;
         x != vec.begin()-1; x--)
        cout << *x << endl;
}
```

Percorrer um vector ao contrário

- Podemos usar funções `rbegin()` e `rend()`
- Reverse begin e reverse end
 - Itera em ordem reversa
- E usar o iterarator normalmente

```
#include <iostream>
#include <vector>
#include <string>

using namespace std;

int main () {
    vector<int> vec {0, 1,
                    1, 2, 3, 5, 8, 13, 21};
    for (auto x = vec.begin();
         x != vec.end(); x++)
        cout << *x << endl;
    cout << "-----" << endl;
    for (auto x = vec.rbegin();
         x != vec.rend(); x++)
        cout << *x << endl;
}
```

Iterator implícito

- Também conhecido como range for
- Declara uma variável
 - Tipo do dado do vector
- Declara um for associando a variável ao vector
 - Itera normalmente
 - Mas o iterator é implícito: não é um ponteiro (cast)

```
#include <iostream>
#include <vector>
#include <string>

using namespace std;

int main () {
    vector<string> vec;
    vec.push_back("Mestre");
    vec.push_back("Zangado");
    vec.push_back("Atchim");
    vec.push_back("Dunga");
    vec.push_back("Soneca");
    vec.push_back("Dengoso");
    vec.push_back("Feliz");
    for (string x: vec)
        cout << x << endl;
}
```