

STRINGS

CADEIA DE CARACTERES

Estruturas de dados

2

- Modo de armazenar e manipular dados na memória de um computador.
- Estruturas de dados homogêneas:
 - ▣ Mesmo tipo de dado agrupado.
 - ▣ Variável indexada (utilizam índices para diferenciar).
 - ▣ Vetores, Strings e Matrizes.
- Estrutura de dados heterogêneas:
 - ▣ Diferentes dados agrupados.
 - ▣ Registros (structs em C).

Conceito de vetor

3

- ❑ Declarar um vetor equivale a reservar um espaço na memória temporariamente.
- ❑ Espaço dividido de acordo com o tipo de dado.
- ❑ Vários valores do mesmo tipo.

- ❑ Agrupamento de variáveis do mesmo tipo, identificadas por índices.

Conceito de vetor

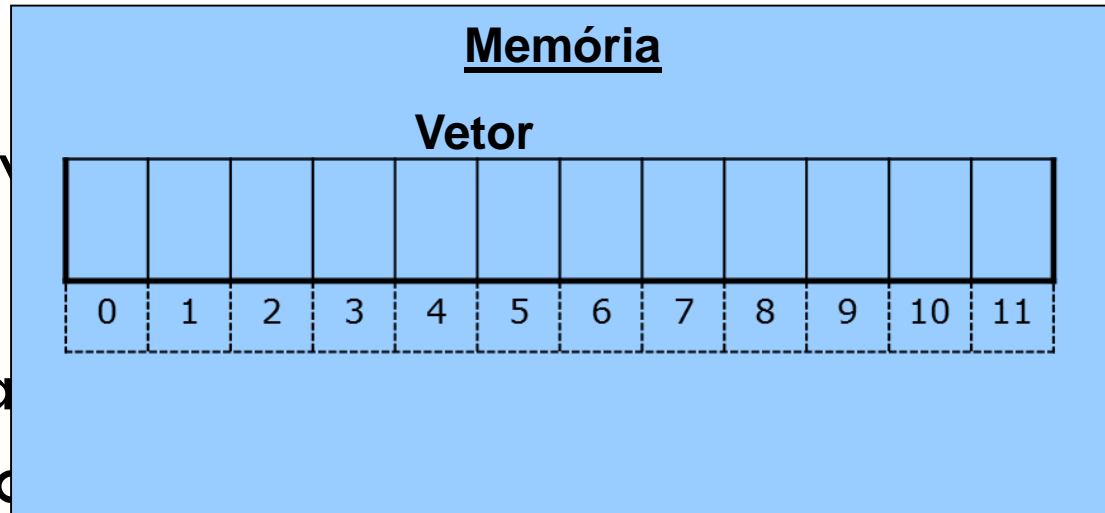
4

- ❑ Declarar um vetor equivale a reservar um espaço na memória temporariamente.

- ❑ Espaço

- ❑ Vários v

- ❑ Agrupa
identific



ado.

Problema

5

□ Armazenar e manipular textos em C.

```
#include <stdio.h>

int main()
{
    char letra0='A', letra1='l', letra2='g', letra3='o', letra4='r', letra5='i';
    char letra6='t', letra7='m', letra8='o', letra9='s', letra10='2', letra11='\n';

    printf("%c%c%c%c%c%c%c%c%c%c%c%c", letra0, letra1, letra2, letra3, letra4,
        letra5, letra6, letra7, letra8, letra9, letra10, letra11);

    return 0;
}
```

Conceito de string

6

- Cadeia de caracteres ou vetor de caracteres (tipo de dado ***char***).

Conceito de string

7

- Cadeia de caracteres ou vetor de caracteres (tipo de dado *char*).
- As strings se diferenciam por ter significado no contexto geral do vetor.

Conceito de string

8

- Cadeia de caracteres ou vetor de caracteres (tipo de dado *char*).
- As strings se diferenciam por ter significado no contexto geral do vetor.
- Cada índice armazena uma letra, porém o conjunto forma a palavra ou frase.

Conceito de string

9

- Cadeia de caracteres ou vetor de caracteres (tipo de dado *char*).
- As strings se diferenciam por ter significado no contexto geral do vetor.
- Cada índice armazena uma letra, porém o conjunto forma a palavra ou frase.
- Utiliza um caractere de controle para **identificar o final do texto**.
 - `'\0'`

Entendendo o char

10

- O tipo **char** é utilizado para representar **caracteres**.

Entendendo o char

11

- ❑ O tipo **char** é utilizado para representar **caracteres**.
- ❑ Um caractere é representado através de um byte na memória.
 - ▣ 1 byte tem 8 bits, ou seja, é possível representar 256 números (ou no caso, codificar até 256 caracteres distintos).

Entendendo o char

12

- ❑ O tipo **char** é utilizado para representar **caracteres**.
- ❑ Um caractere é representado através de um byte na memória.
 - ▣ 1 byte tem 8 bits, ou seja, é possível representar 256 números (ou no caso, codificar até 256 caracteres distintos).
- ❑ A linguagem C utiliza esse número como um índice na tabela ASCII.

Entendendo o char

13

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`	128	80	Ç	160	A0	á	192	C0	À
1	01	Start of heading	33	21	!	65	41	A	97	61	a	129	81	ü	161	A1	í	193	C1	Á
2	02	Start of text	34	22	"	66	42	B	98	62	b	130	82	é	162	A2	ó	194	C2	Â
3	03	End of text	35	23	#	67	43	C	99	63	c	131	83	â	163	A3	ú	195	C3	Ã
4	04	End of transmit	36	24	\$	68	44	D	100	64	d	132	84	ä	164	A4	ñ	196	C4	Ä
5	05	Enquiry	37	25	%	69	45	E	101	65	e	133	85	å	165	A5	Ñ	197	C5	Å
6	06	Acknowledge	38	26	&	70	46	F	102	66	f	134	86	ä	166	A6	*	198	C6	Æ
7	07	Audible bell	39	27	'	71	47	G	103	67	g	135	87	ç	167	A7	º	199	C7	Ç
8	08	Backspace	40	28	(72	48	H	104	68	h	136	88	ê	168	A8	¿	200	C8	È
9	09	Horizontal tab	41	29)	73	49	I	105	69	i	137	89	ë	169	A9	¸	201	C9	É
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j	138	8A	è	170	AA	¬	202	CA	Ê
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k	139	8B	ì	171	AB	½	203	CB	Ë
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l	140	8C	î	172	AC	¾	204	CC	Ï
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m	141	8D	ï	173	AD	¿	205	CD	Ð
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n	142	8E	ÿ	174	AE	«	206	CE	Ñ
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o	143	8F	ÿ	175	AF	»	207	CF	Ò
16	10	Data link escape	48	30	0	80	50	P	112	70	p	144	90	É	176	B0	☐	208	D0	Ó
17	11	Device control 1	49	31	1	81	51	Q	113	71	q	145	91	æ	177	B1	☐	209	D1	Ô
18	12	Device control 2	50	32	2	82	52	R	114	72	r	146	92	æ	178	B2	☐	210	D2	Õ
19	13	Device control 3	51	33	3	83	53	S	115	73	s	147	93	ô	179	B3		211	D3	Ö
20	14	Device control 4	52	34	4	84	54	T	116	74	t	148	94	ö	180	B4		212	D4	×
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u	149	95	ò	181	B5		213	D5	×
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v	150	96	û	182	B6		214	D6	×
23	17	End trans. block	55	37	7	87	57	W	119	77	w	151	97	ù	183	B7		215	D7	×
24	18	Cancel	56	38	8	88	58	X	120	78	x	152	98	ÿ	184	B8		216	D8	×
25	19	End of medium	57	39	9	89	59	Y	121	79	y	153	99	ÿ	185	B9		217	D9	×
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z	154	9A	ÿ	186	BA		218	DA	×
27	1B	Escape	59	3B	;	91	5B	[123	7B	{	155	9B	¢	187	BB		219	DB	☐
28	1C	File separator	60	3C	<	92	5C	\	124	7C		156	9C	£	188	BC		220	DC	☐
29	1D	Group separator	61	3D	=	93	5D]	125	7D	}	157	9D	¥	189	BD		221	DD	☐
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~	158	9E	¥	190	BE		222	DE	☐
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	☐	159	9F	f	191	BF		223	DF	☐

Entendendo o char

14

- Quando uma tecla é digitada (lida pelo *scanf*), o código correspondente à tecla é **traduzido** para o número binário correspondente, e armazenado na variável utilizada no *scanf*.
- Quando uma variável caractere é utilizada no *printf*, esse número é utilizado para imprimir o caractere correspondente na **tabela ASCII**.

Conceito de string

15

- char **Genero**;
- char **Nome**[35];
- char **Texto**[256];

Conceito de string

16

- char **Genero**;
- char **Nome**[35];
- char **Texto**[256];

- **Genero** é uma variável que armazenará apenas um caracter, denotado por apóstrofes ou **aspas simples**.

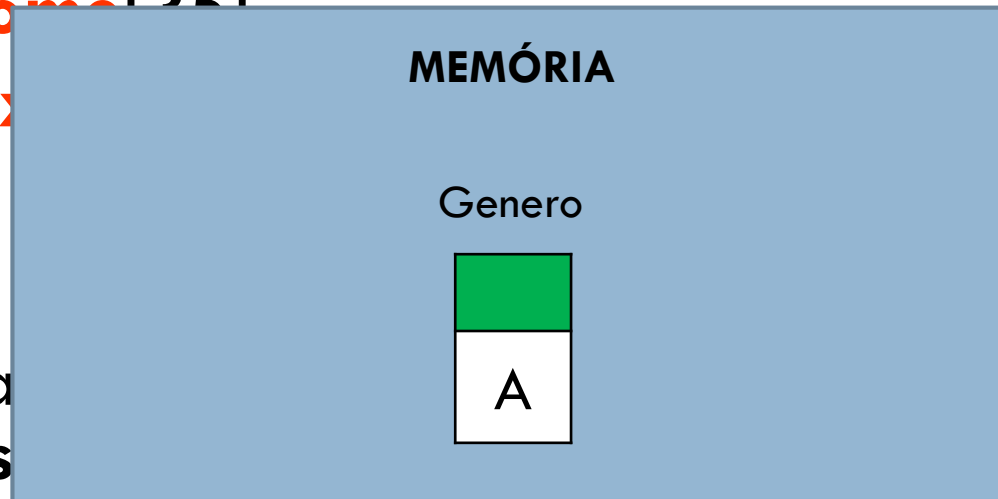
- char **Genero** = 'A'; // aspas simples converte o valor numérico da tabela ASCII

Conceito de string

17

- ❑ char **Genero**;
- ❑ char **Nome**[25];
- ❑ char **Text**;

- ❑ **Genero** apenas um caractere, portanto, precisa de apenas uma aspa simples



- ❑ char **Genero** = 'A'; //aspas simples converte o valor numérico da tabela ASCII

Conceito de string

18

- char **Genero**;
- char **Nome**[35];
- char **Texto**[256];

- **Nome** é uma variável *string* que pode conter no máximo 34 caracteres, indexados de **Nome[0]** até **Nome[34]**.

Conceito de string

19

- char **Genero**;
- char **Nome**[35];
- char **Texto**[256];

- **Nome** é uma variável *string* que pode conter no máximo 34 caracteres, indexados de **Nome[0]** até **Nome[34]**.

- **Nome[34]** conterá o caractere **NULL** (**\0**) para representar o final da string.

Conceito de string

20

- char **Genero**;
- char **Nome**[35];
- char **Texto**[256];

- **Nome** é uma variável *string* que pode conter no máximo 34 caracteres, indexados de **Nome[0]** até **Nome[34]**.
- **Nome[34]** conterá o caractere **NULL** (**\0**) para representar o final da string.
 - ▣ **A validação do limite do vetor é responsabilidade do programador! Nem sempre o compilador adiciona ao final.**

Conceito de string

21

- char **Genero**;
- char **Nome**[35];

MEMÓRIA

Nome

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	...	34
M	u	r	i	e	l	_	M	a	z	z	e	t	t	o	\0	;	'	/

representar o final da string.

- **A validação do limite do vetor é responsabilidade do programador! Nem sempre o compilador adiciona ao final.**

Conceito de string

22

- ❑ char **Genero**;
- ❑ char **Nome**[35];

TEXTO ARMAZENADO DENTRO DA
STRING.

MEMÓRIA

Nome

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	...	34
M	u	r	i	e	l	_	M	a	z	z	e	t	t	o	\0	;	'	/

representar o final da string.

- ❑ **A validação do limite do vetor é responsabilidade do programador! Nem sempre o compilador adiciona ao final.**

Conceito de string

23

- ❑ char **Genero**;
- ❑ char **Nome**[35];

INDICADOR DE **FINAL DO TEXTO**.
ENTRE TEXTO E LIXO DE MEMÓRIA.

MEMÓRIA

Nome

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	...	34
M	u	r	i	e	l	_	M	a	z	z	e	t	t	o	\0	;	'	/

representar o final da string.

- ❑ **A validação do limite do vetor é responsabilidade do programador! Nem sempre o compilador adiciona ao final.**

Conceito de string

24

- ❑ char **Genero**;
- ❑ char **Nome**[35];

POSIÇÕES RESTANTES
PREENCHIDAS COM **LIXO DE
MEMÓRIA.**

MEMÓRIA

Nome

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	...	34
M	u	r	i	e	l	_	M	a	z	z	e	t	t	o	\0	;	'	/

representar o final da string.

- ❑ **A validação do limite do vetor é responsabilidade do programador! Nem sempre o compilador adiciona ao final.**

Conceito de string

25

- char **Genero**;
- char **Nome**[35];
- char **Texto**[256];

- **Texto** é uma *string* capaz de armazenar 256 caracteres, indexados de `Texto[0]` até `Texto[255]`.

Conceito de string

26

- char **Genero**;
- char **Nome**[35];
- char **Texto**[256];

- **Texto** é uma *string* capaz de armazenar 256 caracteres, indexados de `Texto[0]` até `Texto[255]`.

- Em `Texto[255]`, ou após o último caractere **armazendo**, conterá o caractere **NULL** (`\0`) para representar o final da string.

Questionário

27

- 1 - Defina o que é uma string.
- 2 - Quais as diferenças de uma string para um vetor?
- 3 - Declare uma string em C para armazenar uma palavra de 25 letras.

Lendo e escrevendo char

28

- Por padrão, a leitura e a escrita de char utilizam o comando de controle “%c” nas funções de entrada e saída.
 - ▣ `scanf(“%c”, &Letra);`
 - ▣ `printf(“%c”, Letra);`
- Possui funções próprias:
 - ▣ `Letra = getchar();`
 - ▣ `putchar(Letra);`

Lendo e escrevendo string

29

- Comando “%s” nas funções de entrada e saída para strings.
 - char Palavra[20];
 - scanf(“%s”, Palavra);
 - printf(“%s”, Palavra);
- Possui funções próprias:
 - gets(Palavra);
 - puts(Palavra);

Lendo e escrevendo string

30

- Comando “%s” nas funções de entrada e saída para strings.

- char Palavra[20];

- scanf(“%s”, Palavra);

- printf(“%s”, Palavra);

- Possui funções próprias:

- gets(Palavra);

- puts(Palavra);



- NÃO SE UTILIZA & PARA
LEITURA DE STRING

Lendo e escrevendo string

31

- Comando “%s” nas funções de entrada e saída para strings.

- char Palavra[20];

- scanf(“%s”, Palavra);

- printf(“%s”, Palavra);

- Possui funções próprias:

- gets(Palavra);

- puts(Palavra);



- NÃO SE UTILIZA & PARA
LEITURA DE STRING

- NÃO SE UTILIZA ÍNDICE
QUANDO ESTÁ
LENDO/ESCREVENDO TODA A
PALAVRA

Lendo string

32

- `scanf("%s", Palavra);`
 - ▣ Faz a leitura dos caracteres até ler um **ESPAÇO** ou **ENTER**.

Lendo string

33

- `scanf("%s", Palavra);`
 - ▣ Faz a leitura dos caracteres até ler um **ESPAÇO** ou **ENTER**.

- `gets(Palavra);`
 - ▣ Faz a leitura dos caracteres até ler um **ENTER**.

Lendo string

34

- `scanf("%s", Palavra);`
 - ▣ Faz a leitura dos caracteres até ler um **ESPAÇO** ou **ENTER**.

- `gets(Palavra);`
 - ▣ Faz a leitura dos caracteres até ler um **ENTER**.

- `scanf("%[^\n]s", Palavra);`
 - ▣ Faz a leitura dos caracteres até ler um **ENTER**.

Lendo string

35

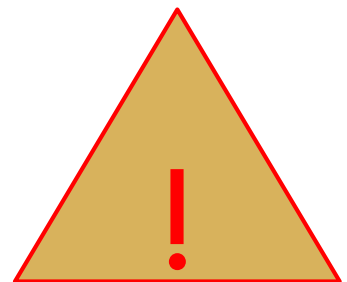
```
int main(void)
{
    char palavra[20];

    { //Leitura até ESPAÇO ou ENTER
      scanf("%s", palavra);
      printf("%s\n", palavra);
    }

    { //Leitura até ENTER
      scanf("%[^\n]s", palavra);
      printf("%s\n", palavra);
    }

    { //Leitura até ENTER
      gets(palavra);
      puts(palavra);
    }

    return 0;
}
```



Lendo string

36

- Problema de leituras consecutivas:
 - ▣ A leitura utiliza um buffer do teclado.

Lendo string

37

- Problema de leituras consecutivas:
 - ▣ A leitura utiliza um buffer do teclado.
 - ▣ O buffer armazena os últimos caracteres lidos.

Lendo string

38

- Problema de leituras consecutivas:
 - ▣ A leitura utiliza um buffer do teclado.
 - ▣ O buffer armazena os últimos caracteres lidos.
 - ▣ Ao realizar uma nova leitura, o código busca o que está dentro do buffer, podendo pegar **resíduo** da leitura anterior (um enter ou espaço).

Lendo string

39

- Problema de leituras consecutivas:
 - ▣ A leitura utiliza um buffer do teclado.
 - ▣ O buffer armazena os últimos caracteres lidos.
 - ▣ Ao realizar uma nova leitura, o código busca o que está dentro do buffer, podendo pegar **resíduo** da leitura anterior (um enter ou espaço).
- Correção 1:
 - ▣ Utilizando **apenas scanf()**: utilizar um espaço ante do %s. Ex: `scanf(" %s", palavra);`

Lendo string

40

- Problema de leituras consecutivas:
 - ▣ A leitura utiliza um buffer do teclado.
 - ▣ O buffer armazena os últimos caracteres lidos.
 - ▣ Ao realizar uma nova leitura, o código busca o que está dentro do buffer, podendo pegar **resíduo** da leitura anterior (um enter ou espaço).
- Correção 2:
 - ▣ Utilizando **diferentes funções de leitura**: limpeza de buffer antes de leitura de novos dados.
 - `fflush(stdin); //windows`
 - `__fpurge(stdin); //linux`

Lendo string

41

```
int main(void)
{
    char palavra[20];

    //Leitura até ESPAÇO ou ENTER
    scanf("%s", palavra);
    printf("%s\n", palavra);

    //Leitura até ENTER
    scanf(" %[^\n]s", palavra);
    printf("%s\n", palavra);

    //Leitura até ENTER
    fflush(stdin); // __fpurge(stdin);
    gets(palavra);
    puts(palavra);

    return 0;
}
```

Questionário

42

- 4 - Escreva um trecho de código que:
 - ▣ Declare uma string de tamanho 20;
 - ▣ Leia uma palavra do teclado e imprima o que foi lido, três vezes.

Lendo string

43

```
int main(void)
{
    char palavra[20];
    int i, N = 3;

    for(i = 0; i < N; i++){
        //Leitura até ESPAÇO ou ENTER
        scanf(" %s", palavra);
        printf("%s\n", palavra);
    }

    for(i = 0; i < N; i++){
        //Leitura até ENTER
        scanf("%[^\n]s", palavra);
        printf("%s\n", palavra);
    }

    for(i = 0; i < N; i++){
        //Leitura até ENTER
        fflush(stdin); //__fpurge(stdin);
        gets(palavra);
        puts(palavra);
    }

    return 0;
}
```

Inicialização

44

```
#include <stdio.h>

int main(void)
{
    //Forma Convencional: programador deve incluir \0 ao final
    char nome1[20] = {'A', 'L', 'G', 'O', 'R', 'I', 'T', 'M', 'O', 'S', '2', '\0'};

    //Forma Especial: compilador inclui \0 ao final
    char nome2[20] = "ALGORITMOS2";

    printf("%s \n", nome1);
    printf("%s \n", nome2);

    return 0;
}
```

Manipulação

45

```
#include <stdio.h>

int main(void)
{
    char palavra[15] = "ALGORITMOS";

    printf("%s \n", palavra);

    //Alterar letra por letra
    //Utilizando os índices separadamente
    palavra[0] = 'b';
    palavra[1] = 'o';
    palavra[2] = 'l';
    palavra[3] = 'o';

    printf("%s \n", palavra);

    return 0;
}
```

Manipulação

46

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

```
int main(void)
{
```

```
    char palavra[15] = "ALGORITMOS";
```

```
    printf("%s \n", palavra);
```

```
    //Alterar letra por letra
```


```
    //Utilizando os índices separadamente
```

```
    palavra[0] = 'b';
```

```
    palavra[1] = 'o';
```

```
    palavra[2] = 'l';
```

```
    palavra[3] = 'o';
```

```
 palavra[4] = '\0'; // forçará o final da string aqui
```

```
    //imprimirá apenas "bolo"
```

```
    //ignorando o restante
```

```
    printf("%s \n", palavra);
```

```
    return 0;
```

```
}
```

Manipulação

47

```
#include <stdio.h>

int main(void)
{
    char palavra[15] = "ALGORITMOS";

    printf("%s \n", palavra);

    //Alterar letra por letra
    //Utilizando os índices separadamente
    palavra[0] = 'b';
    palavra[1] = 'o';
    palavra[2] = 'l';
    palavra[3] = 'o';
    palavra[4] = '\\0';// forçará o final da string aqui

    //imprimirá apenas "bolo"
    //ignorando o restante
    printf("%s \n", palavra);

    return 0;
}
```

MEMÓRIA

Manipulação

48

```
#include <stdio.h>

int main(void)
{
    → char palavra[15] = "ALGORITMOS";

    printf("%s \n", palavra);

    //Alterar letra por letra
    //Utilizando os índices separadamente
    palavra[0] = 'b';
    palavra[1] = 'o';
    palavra[2] = 'l';
    palavra[3] = 'o';
    palavra[4] = '\\0'; // forçará o final da string aqui

    //imprimirá apenas "bolo"
    //ignorando o restante
    printf("%s \n", palavra);

    return 0;
}
```

MEMÓRIA

Manipulação

49

```
#include <stdio.h>

int main(void)
{
    → char palavra[15] = "ALGORITMOS";

    printf("%s \n", palavra);

    //Alterar letra por letra
    //Utilizando os índices separadamente
    palavra[0] = 'b';
    palavra[1] = 'o';
    palavra[2] = 'l';
    palavra[3] = 'o';
    palavra[4] = '\\0';// forçará o final da string aqui

    //imprimirá apenas "bolo"
    //ignorando o restante
    printf("%s \n", palavra);

    return 0;
}
```

[illegible]

Manipulação

50

```
#include <stdio.h>

int main(void)
{
    → char palavra[15] = "ALGORITMOS";

    printf("%s \n", palavra);

    //Alterar letra por letra
    //Utilizando os índices separadamente
    palavra[0] = 'b';
    palavra[1] = 'o';
    palavra[2] = 'l';
    palavra[3] = 'o';
    palavra[4] = '\\0'; // forçará o final da string aqui

    //imprimirá apenas "bolo"
    //ignorando o restante
    printf("%s \n", palavra);

    return 0;
}
```

MEMÓRIA

palavra

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	L	G	O	R	I	T	M	O	S	\0				

Manipulação

51

```
#include <stdio.h>

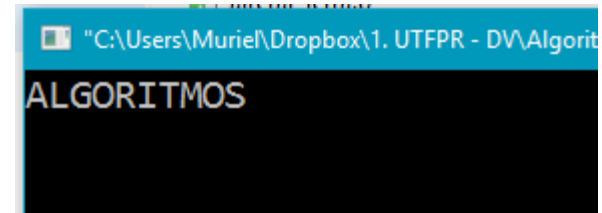
int main(void)
{
    char palavra[15] = "ALGORITMOS";

    → printf("%s \n", palavra);

    //Alterar letra por letra
    //Utilizando os índices separadamente
    palavra[0] = 'b';
    palavra[1] = 'o';
    palavra[2] = 'l';
    palavra[3] = 'o';
    palavra[4] = '\\0'; // forçará o final da string aqui

    //imprimirá apenas "bolo"
    //ignorando o restante
    printf("%s \n", palavra);

    return 0;
}
```



MEMÓRIA														
palavra														
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	L	G	O	R	I	T	M	O	S	\\0				

Manipulação

52

```
#include <stdio.h>

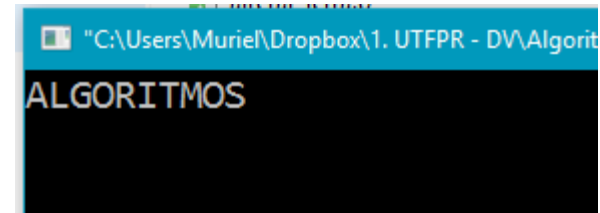
int main(void)
{
    char palavra[15] = "ALGORITMOS";

    printf("%s \n", palavra);

    //Alterar letra por letra
    //Utilizando os índices separadamente
    → palavra[0] = 'b';
    palavra[1] = 'l';
    palavra[2] = 'g';
    palavra[3] = 'o';
    palavra[4] = '\0'; // forçará o final da string aqui

    //imprimirá apenas "bolo"
    //ignorando o restante
    printf("%s \n", palavra);

    return 0;
}
```



MEMÓRIA														
palavra														
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
b	L	G	O	R	I	T	M	O	S	\0				

Manipulação

53

```
#include <stdio.h>

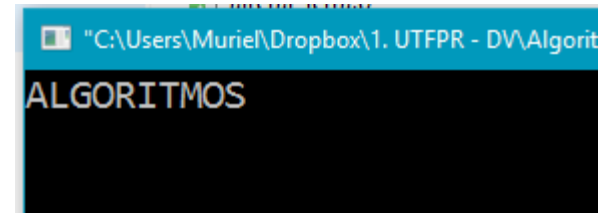
int main(void)
{
    char palavra[15] = "ALGORITMOS";

    printf("%s \n", palavra);

    //Alterar letra por letra
    //Utilizando os índices separadamente
    palavra[0] = 'b';
    → palavra[1] = 'o';
    palavra[2] = 'l';
    palavra[3] = 'o';
    palavra[4] = '\\0'; // forçará o final da string aqui

    //imprimirá apenas "bolo"
    //ignorando o restante
    printf("%s \n", palavra);

    return 0;
}
```



MEMÓRIA														
palavra														
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
b	o	G	O	R	I	T	M	O	S	\\0				

Manipulação

54

```
#include <stdio.h>

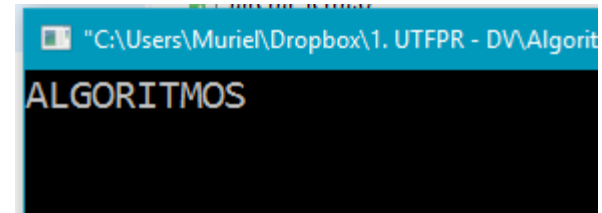
int main(void)
{
    char palavra[15] = "ALGORITMOS";

    printf("%s \n", palavra);

    //Alterar letra por letra
    //Utilizando os índices separadamente
    palavra[0] = 'b';
    palavra[1] = 'o';
    → palavra[2] = 'l';
    palavra[3] = 'o';
    palavra[4] = '\\0'; // forçará o final da string aqui

    //imprimirá apenas "bolo"
    //ignorando o restante
    printf("%s \n", palavra);

    return 0;
}
```



MEMÓRIA														
palavra														
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
b	o	l	O	R	I	T	M	O	S	\\0				

Manipulação

55

```
#include <stdio.h>

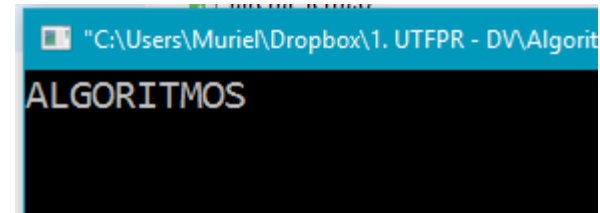
int main(void)
{
    char palavra[15] = "ALGORITMOS";

    printf("%s \n", palavra);

    //Alterar letra por letra
    //Utilizando os índices separadamente
    palavra[0] = 'b';
    palavra[1] = 'o';
    palavra[2] = 'l';
    → palavra[3] = 'o';
    palavra[4] = '\\0';// forçará o final da string aqui

    //imprimirá apenas "bolo"
    //ignorando o restante
    printf("%s \n", palavra);

    return 0;
}
```



MEMÓRIA														
palavra														
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
b	o	l	o	R	I	T	M	O	S	\\0				

Manipulação

56

```
#include <stdio.h>

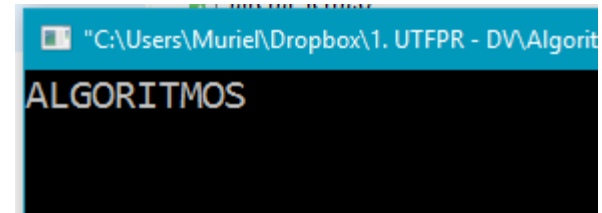
int main(void)
{
    char palavra[15] = "ALGORITMOS";

    printf("%s \n", palavra);

    //Alterar letra por letra
    //Utilizando os índices separadamente
    palavra[0] = 'b';
    palavra[1] = 'o';
    palavra[2] = 'l';
    palavra[3] = 'o';
    → palavra[4] = '\0'; // forçará o final da string aqui

    //imprimirá apenas "bolo"
    //ignorando o restante
    printf("%s \n", palavra);

    return 0;
}
```



MEMÓRIA														
palavra														
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
b	o	l	o	\0	l	T	M	O	S	\0				

Manipulação

57

```
#include <stdio.h>

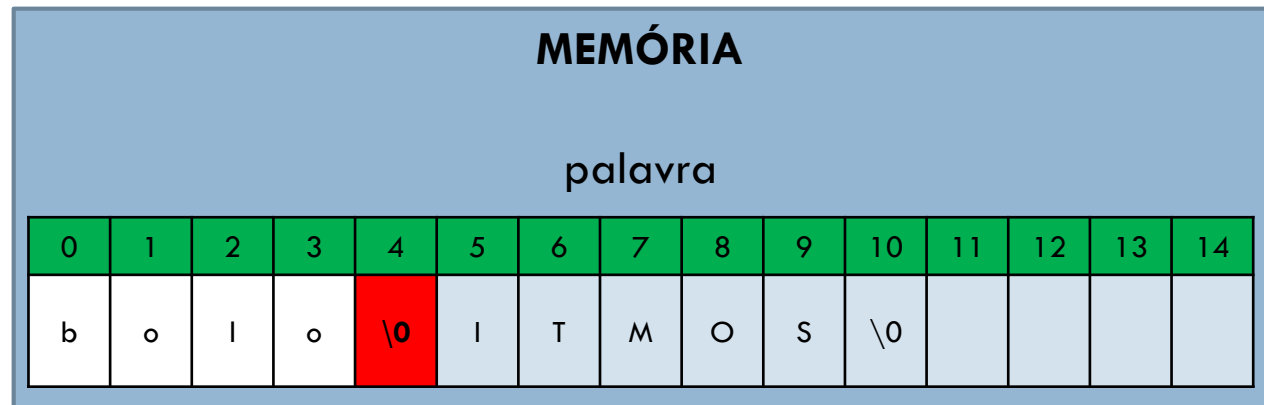
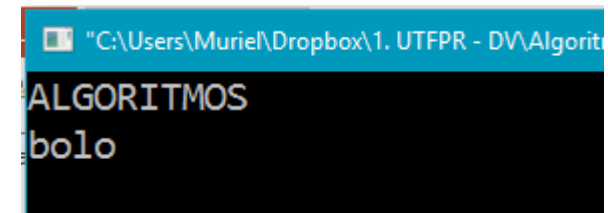
int main(void)
{
    char palavra[15] = "ALGORITMOS";

    printf("%s \n", palavra);

    //Alterar letra por letra
    //Utilizando os índices separadamente
    palavra[0] = 'b';
    palavra[1] = 'o';
    palavra[2] = 'l';
    palavra[3] = 'o';
    palavra[4] = '\\0'; // forçará o final da string aqui

    //imprimirá apenas "bolo"
    //ignorando o restante
    → printf("%s \n", palavra);

    return 0;
}
```



Manipulação

58

```
#include <stdio.h>

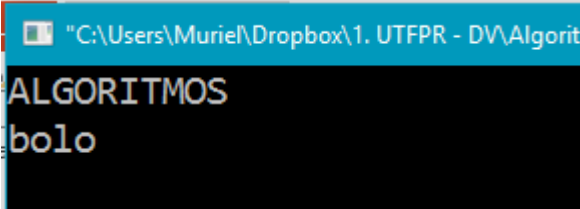
int main(void)
{
    char palavra[15] = "ALGORITMOS";

    printf("%s \n", palavra);

    //Alterar letra por letra
    //Utilizando os índices separadamente
    palavra[0] = 'b';
    palavra[1] = 'o';
    palavra[2] = 'l';
    palavra[3] = 'o';
    palavra[4] = '\\0';// forçará o final da string aqui

    //imprimirá apenas "bolo"
    //ignorando o restante
    printf("%s \n", palavra);

    → return 0;
}
```



```
"C:\Users\Muriel\Dropbox\1. UTFPR - DV\Algoritmos"
ALGORITMOS
bolo
```

FIM DA EXECUÇÃO

MEMÓRIA

palavra

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
b	o	l	o	\0	l	T	M	O	S	\0				

Copiar valores

59

- Para atribuir um valor de uma variável em outra, basta usar atribuição simples:

- char L_1 = 'a';

- char L_2 = L_1;

Copiar valores

60

- Para atribuir um valor de uma variável em outra, basta usar atribuição simples:
 - `char L_1 = 'a';`
 - `char L_2 = L_1;`
- **Porém, para copiar vetores é necessário copiar índice por índice.** Cada qual com sua respectiva posição.

Copiar valores

61

```
int main(void)
{
    char palavra1[15] = "ALGORITMOS";
    char palavra2[15]; //Não inicializada

    int i;

    //Continuará o loop até encontrar \0
    for(i = 0; palavra1[i]!='\0'; i++)
    {
        palavra2[i] = palavra1[i];
    }

    //O \0 não é copiado por ser o critério de parada.
    //NECESSÁRIO INCLUIR O \0 NO FINAL DA PALAVRA COPIADA.
    //A variável i já está na posição quando sai do loop.
    palavra2[i] = '\0';

    printf("%s == %s", palavra1, palavra2);

    return 0;
}
```

Copiar valores

62

```
int main(void)
{
    char palavra1[15] = "ALGORITMOS";
    char palavra2[15]; //Não inicializada

    int i;

    //Continuará o loop até encontrar \0
    for(i = 0; palavra1[i]!='\0'; i++)
    {
        palavra2[i] = palavra1[i];
    }

    //O \0 não é copiado por ser o critério de parada.
    //NECESSÁRIO INCLUIR O \0 NO FINAL DA PALAVRA COPIADA.
    //A variável i já está na posição quando sai do loop.
    palavra2[i] = '\0';

    printf("%s == %s", palavra1, palavra2);

    return 0;
}
```

MEMÓRIA

Copiar valores

63

```
int main(void)
{
→ char palavra1[15] = "ALGORITMOS";
  char palavra2[15]; //Não inicializada

  int i;
```

DECLARAÇÃO E
INICIALIZAÇÃO DAS
VARIÁVEIS

MEMÓRIA

64

```
→ char palavra1[15] = "ALGORITMOS";  
   char palavra2[15]; //Não inicializada
```

MEMÓRIA

[illegible]

Copiar valores

65

```
int main(void)
{
→ char palavra1[15] = "ALGORITMOS";
  char palavra2[15]; //Nao inicializada

  int i;
```

DECLARAÇÃO E
INICIALIZAÇÃO DAS
VARIÁVEIS

MEMÓRIA

palavra1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	L	G	O	R	I	T	M	O	S	\0				

66

```
➡ char palavra2[15]; //Não inicializada
```

MEMÓRIA

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	L	G	O	R	I	T	M	O	S	\0				

[illegible]

67

→ `int i;`

MEMÓRIA

i

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	L	G	O	R	I	T	M	O	S	\0				

[illegible]

68

```
→ for(i = 0; palavra1[i]!='\0'; i++)
{
    palavra2[i] = palavra1[i];
}
```

MEMÓRIA

i

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	L	G	O	R	I	T	M	O	S	\0				

[illegible]

69

i COMEÇA EM 0

i **0**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	L	G	O	R	I	T	M	O	S	\0				

[illegible]

70

CONTINUA SE
palavra1[0] É
DIFERENTE DE '\0'

i	0
---	---

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	L	G	O	R	I	T	M	O	S	\0				

[illegible]

71

COPIAR CHAR DO ÍNDICE PARA OUTRA STRING

[illegible]

72

```
→ for(i = 0; palavra1[i]!='\0'; i++)
{
    palavra2[i] = palavra1[i];
}
```

MEMÓRIA



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	L	G	O	R	I	T	M	O	S	\0				

[illegible]

73

```
→ for(i = 0; palavra1[i] != '\0'; i++)
{
    palavra2[i] = palavra1[i];
}
```

MEMÓRIA

i 1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	L	G	O	R	I	T	M	O	S	\0				

[illegible]

74

COPIAR CHAR DO ÍNDICE PARA OUTRA STRING

MEMÓRIA

palavra1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	L	G	O	R	I	T	M	O	S	\0				

palavra2

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	L													

The diagram illustrates memory layout. It shows two memory words, 'palavra1' and 'palavra2', each with 15 slots (indices 0 to 14). 'palavra1' contains the string 'ALGORITMOS' followed by a null terminator '\0' at index 10. 'palavra2' contains the string 'AL' followed by null terminators. A red box highlights the first two slots (indices 0 and 1) of both words, indicating the start of the string storage.

75

INCREMENTO DE i
OCORRE **APÓS** O
CONTEÚDO DO
LOOP

[illegible]

76

CONTINUA SE
palavra1[2] É
DIFERENTE DE '\0'

MEMÓRIA

palavra1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	L	G	O	R	I	T	M	O	S	\0				

palavra2

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	L													

The diagram illustrates memory layout. It shows two memory blocks, 'palavra1' and 'palavra2', each with 15 slots (indices 0 to 14). 'palavra1' contains the string 'ALGORITMOS' followed by a null terminator '\0'. 'palavra2' contains the string 'AL' followed by empty slots. A red box highlights the memory slot at index 2 of 'palavra1', which contains the character 'G'. In the top right corner, there is a small box with 'i' and a red '2'.

77

COPIAR CHAR DO ÍNDICE PARA OUTRA STRING

MEMÓRIA

palavra1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	L	G	O	R	I	T	M	O	S	\0				

palavra2

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	L	G												

The diagram illustrates memory allocation for two strings, "palavra1" and "palavra2". Each string is represented by an array of 15 slots (indices 0 to 14). The first row shows "palavra1" containing the characters A, L, G, O, R, I, T, M, O, S, followed by a null terminator (\0) at index 10. The second row shows "palavra2" containing the characters A, L, G, followed by null terminators from index 3 onwards. A red box highlights the third slot (index 2) in both arrays, indicating the current position being processed.

78

INCREMENTO DE i
OCORRE **APÓS** O
CONTEÚDO DO
LOOP

[illegible]

79



CONTINUA SE
palavra1[3] É
DIFERENTE DE ‘\0’

MEMÓRIA

i	3
---	---

palavra1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	L	G	O	R	I	T	M	O	S	\0				

palavra2

[illegible]

80

COPIAR CHAR DO ÍNDICE PARA OUTRA STRING

[illegible]

81

```
→ for(i = 0; palavra1[i]!='\0'; i++)
{
    palavra2[i] = palavra1[i];
}
```

MEMÓRIA

i 4

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	L	G	O	R	I	T	M	O	S	\0				

[illegible]

82

i	4
---	---

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	L	G	O	R	I	T	M	O	S	\0				

[illegible]

83

→ palavra2[i] = palavra1[i];

i	4
---	---

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	L	G	O	R	I	T	M	O	S	\0				

[illegible]

84

```
→ for(i = 0; palavra1[i]!='\0'; i++)
{
    palavra2[i] = palavra1[i];
}
```

```
palavra2[i] = palavra1[i];
```

i 5

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	L	G	O	R	I	T	M	O	S	\0				

[illegible]

85

```
→ for(i = 0; palavra1[i] != '\0'; i++)
{
    palavra2[i] = palavra1[i];
}
```

i	5
---	---

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	L	G	O	R	I	T	M	O	S	\0				

[illegible]

Copiar valores

89

```
//Continuará o loop até encontrar \0  
for(i = 0; palavra1[i]!='\0'; i++)  
{  
    → palavra2[i] = palavra1[i];  
}
```

MEMÓRIA

i 6

palavra1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	L	G	O	R	I	T	M	O	S	\0				

palavra2

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	L	G	O	R	I	T								

Copiar valores

90

```
→ //Continuará o loop até encontrar \0  
for(i = 0; palavra1[i] != '\0'; i++)  
{  
    palavra2[i] = palavra1[i];  
}
```

MEMÓRIA

i 7

palavra1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	L	G	O	R	I	T	M	O	S	\0				

palavra2

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	L	G	O	R	I	T								

Copiar valores

91

```
→ //Continuará o loop até encontrar \0  
for(i = 0; palavra1[i] != '\0'; i++)  
{  
    palavra2[i] = palavra1[i];  
}
```

MEMÓRIA

i 7

palavra1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	L	G	O	R	I	T	M	O	S	\0				

palavra2

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	L	G	O	R	I	T								

Copiar valores

92

```
//Continuará o loop até encontrar \0  
for(i = 0; palavra1[i]!='\0'; i++)  
{  
    → palavra2[i] = palavra1[i];  
}
```

MEMÓRIA

i 7

palavra1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	L	G	O	R	I	T	M	O	S	\0				

palavra2

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	L	G	O	R	I	T	M							

Copiar valores

93

```
→ //Continuará o loop até encontrar \0  
for(i = 0; palavra1[i] != '\0'; i++)  
{  
    palavra2[i] = palavra1[i];  
}
```

MEMÓRIA

i 8

palavra1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	L	G	O	R	I	T	M	O	S	\0				

palavra2

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	L	G	O	R	I	T	M							

Copiar valores

94

```
→ //Continuará o loop até encontrar \0  
for(i = 0; palavra1[i] != '\0'; i++)  
{  
    palavra2[i] = palavra1[i];  
}
```

MEMÓRIA

i 8

palavra1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	L	G	O	R	I	T	M	O	S	\0				

palavra2

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	L	G	O	R	I	T	M							

Copiar valores

95

```
//Continuará o loop até encontrar \0  
for(i = 0; palavra1[i]!='\0'; i++)  
{  
    → palavra2[i] = palavra1[i];  
}
```

MEMÓRIA

i 8

palavra1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	L	G	O	R	I	T	M	O	S	\0				

palavra2

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	L	G	O	R	I	T	M	O						

Copiar valores

96

```
→ //Continuará o loop até encontrar \0  
for(i = 0; palavra1[i] != '\0'; i++)  
{  
    palavra2[i] = palavra1[i];  
}
```

MEMÓRIA

i 9

palavra1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	L	G	O	R	I	T	M	O	S	\0				

palavra2

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	L	G	O	R	I	T	M	O						

Copiar valores

97

```
→ //Continuará o loop até encontrar \0  
for(i = 0; palavra1[i] != '\0'; i++)  
{  
    palavra2[i] = palavra1[i];  
}
```

MEMÓRIA

i 9

palavra1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	L	G	O	R	I	T	M	O	S	\0				

palavra2

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	L	G	O	R	I	T	M	O						

Copiar valores

98

```
//Continuará o loop até encontrar \0  
for(i = 0; palavra1[i]!='\0'; i++)  
{  
    → palavra2[i] = palavra1[i];  
}
```

MEMÓRIA

i 9

palavra1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	L	G	O	R	I	T	M	O	S	\0				

palavra2

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	L	G	O	R	I	T	M	O	S					

Copiar valores

99

```
→ //Continuará o loop até encontrar \0  
for(i = 0; palavra1[i] != '\0'; i++)  
{  
    palavra2[i] = palavra1[i];  
}
```

MEMÓRIA

i 10

palavra1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	L	G	O	R	I	T	M	O	S	\0				

palavra2

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	L	G	O	R	I	T	M	O	S					

Copiar valores

100

```
→ //Continuará o loop até encontrar \0  
for(i = 0; palavra1[i] != '\0'; i++)  
{  
    palavra2[i] = palavra1[i];  
}
```

MEMÓRIA

i 10

palavra1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	L	G	O	R	I	T	M	O	S	\0				

palavra2

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	L	G	O	R	I	T	M	O	S					

Copiar valores

101

```
→ //Continuará o loop até encontrar \0  
for(i = 0; palavra1[i] != '\0'; i++)  
{  
    palavra2[i] = palavra1[i];  
}
```

- *palavra1[10]* É '\0'
- PARAR LOOP

MEMÓRIA

i	10
---	----

palavra1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	L	G	O	R	I	T	M	O	S	\0				

palavra2

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	L	G	O	R	I	T	M	O	S					

Copiar valores

102

```
//O \0 não é copiado por ser o critério de parada.  
//NECESSÁRIO INCLUIR O \0 NO FINAL DA PALAVRA COPIADA.  
//A variável i já está na posição quando sai do loop.  
→ palavra2[i] = '\0';
```

```
printf("%s == %s", palavra1, palavra2);
```

```
return 0;
```

```
}
```

**- INSERÇÃO DO '\0'
NA POSIÇÃO i**

MEMÓRIA

i	10
---	----

palavra1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	L	G	O	R	I	T	M	O	S	\0				

palavra2

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	L	G	O	R	I	T	M	O	S	\0				

Copiar valores

103

```
//O \0 não é copiado por ser o critério de parada.  
//NECESSÁRIO INCLUIR O \0 NO FINAL DA PALAVRA COPIADA.  
//A variável i já está na posição quando sai do loop.  
palavra2[i] = '\0';
```

```
→ printf("%s == %s", palavra1, palavra2);  
  
return 0;  
}
```

```
C:\Users\Muriele\Dropbox\1. UTFPR - DV\Algoritmos 2\Reso  
ALGORITMOS == ALGORITMOS  
Process returned 0 (0x0)   execu  
Press any key to continue.
```

**- IMPRESSÃO DOS
CARACTERES ANTES
DE '\0'**

MEMÓRIA

i	10
---	----

palavra1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	L	G	O	R	I	T	M	O	S	\0				

palavra2

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	L	G	O	R	I	T	M	O	S	\0				

Questionário

104

- 5 - Explique o que é a inicialização de uma variável.
- 6 - Explique as formas de inicializar uma string em C.
- 7 - Descreva como é possível copiar o conteúdo de um vetor para outro.

Biblioteca para manipulação

105

- Da biblioteca `<string.h>`
 - ▣ **`strcpy(s1,s2)`** – copia `s2` em `s1`
 - ▣ **`strcat(s1,s2)`** – concatena `s2` ao final de `s1`
 - ▣ **`strlen(s1)`** – retorna o tamanho de `s1`
 - ▣ **`strcmp(s1,s2)`** – retorna 0 se `s1` e `s2` são iguais; menor que 0 se `s1 < s2`; maior que 0 se `s1 > s2`
 - ▣ **`strchr(s1,ch)`** – retorna um ponteiro para a primeira ocorrência de `ch` (`char`) em `s1`
 - ▣ **`strstr(s1,s2)`** – retorna um ponteiro para a primeira ocorrência de `s2` em `s1`.

strlen(str)

106

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

int main(void)
{
    char string1[20];
    char string2[20];
    int tamanho = 0;

    //Ler palavra
    scanf("%[^\\n]s", string1);
    scanf(" %[^\\n]s", string2);

    //Função que conta quantidade de caracteres
    tamanho = strlen(string1);
    printf("%s tem %d letras.\\n\\n", string1, tamanho);

    return 0;
}
```

strcpy(str1, str2)

107

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

int main(void)
{
    char string1[20];
    char string2[20];
    int tamanho = 0;

    //Ler palavra
    scanf("%[^\\n]s", string1);
    scanf(" %[^\\n]s", string2);

    //Função que copia string2 EM string1
    strcpy(string1, string2);
    printf("str1: %s \\nstr2: %s \\n\\n", string1, string2);

    return 0;
}
```

strcat(str1, str2)

108

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

int main(void)
{
    char string1[20];
    char string2[20];
    int tamanho = 0;

    //Ler palavra
    scanf("%[^\\n]s", string1);
    scanf(" %[^\\n]s", string2);

    //Função que concatena string2 no final de string1
    strcat(string1, string2);
    printf("%s \\n\\n", string1);

    return 0;
}
```

strcmp(str1, str2)

109

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

int main(void)
{
    char string1[20];
    char string2[20];
    int tamanho = 0;

    //Ler palavra
    scanf("%[^\\n]s", string1);
    scanf(" %[^\\n]s", string2);

    //Função que compara duas strings
    //(Valor de acordo com a ASCII)
    if(strcmp(string1, string2) == 0)
        printf("As palavras sao iguais.\\n\\n");
    else if(strcmp(string1, string2) < 0)
        printf("%s < %s \\n\\n", string1, string2);
    else
        printf("%s > %s \\n\\n", string1, string2);

    return 0;
}
```

Exercícios

110

- 1) Faça um código que:
 - ▣ Leia uma string;
 - ▣ Verifique seu tamanho;
 - ▣ Imprima a string invertida;

- 2) Faça um código que:
 - ▣ Leia uma string;
 - ▣ Conte quantas letras 'a' ou 'A' existem na string;
 - ▣ Imprima a quantidade de vogais;