Caracteres

Conceitos

- Tipo de dados simples (*char*), cujo valor é a codificação numérica de um caracter;
- Caracteres literais são representados por aspas simples, como em ‘A’ e ‘a’;
- Variáveis do tipo char podem receber valores literais do tipo caracter ou também valores inteiros (que nesse caso representam o caracter corresponde, conforme o sistema de codificação adotado);
- Variáveis do tipo char podem também ter o seu valor comparado com inteiros;
- Variáveis do tipo inteiro também poder ser usadas como se fossem do tipo caracter.

Exemplos

```
char c;
int i=67;
...
c='A';
c=c+1;
if (c=='B') ...
c=i;
i=c;
i++;
if (i>='C') ...
```

Codificação

- ASCII – American Standard Code for Information Interchange ;
- Utiliza 7 bits;
- 128 símbolos (caracteres)
- Caracteres visíveis (94), invisíveis (33) e espaço em branco;
- http://pt.wikipedia.org/wiki/ASCII
- Outros sistemas de codificação:
  - EBCDIC (antigo):
    - http://en.wikipedia.org/wiki/Extended_Binary_Coded_Decimal_Interchange_Code
  - Unicode (moderno):
    - http://pt.wikipedia.org/wiki/Unicode
    - http://unicode.org/

Codificação

- O sistema ASCII foi substituído, nos dias de hoje, pelo sistema UTF-8 (8-bit Unicode Transformation Unit);
- UFT-8 e compatível com ASCII na faixa 0 a 127;
- http://en.wikipedia.org/wiki/UTF-8

Binário	Decimal	Hexa	Glifo
0010 0000	32	20	
0010 0001	33	21	!
0010 0010	34	22	"
0010 0011	35	23	#
0010 0100	36	24	$
0010 0101	37	25	%
0010 0110	38	26	&
0010 0111	39	27	'
0010 1000	40	28	(
0010 1001	41	29	)
0010 1010	42	2A	*
0010 1011	43	2B	+
0010 1100	44	2C	,
0010 1101	45	2D	-
0010 1110	46	2E	.
0010 1111	47	2F	/
0011 0000	48	30	0
0011 0001	49	31	1
0011 0010	50	32	2
0011 0011	51	33	3
0011 0100	52	34	4
0011 0101	53	35	5
0011 0110	54	36	6
0011 0111	55	37	7
0011 1000	56	38	8
0011 1001	57	39	9
0011 1010	58	3A	:
0011 1011	59	3B	;
0011 1100	60	3C	<
0011 1101	61	3D	=
0011 1110	62	3E	>
0011 1111	63	3F	?

Binário	Decimal	Hexa	Glifo
0100 0000	64	40	@
0100 0001	65	41	A
0100 0010	66	42	B
0100 0011	67	43	C
0100 0100	68	44	D
0100 0101	69	45	E
0100 0110	70	46	F
0100 0111	71	47	G
0100 1000	72	48	H
0100 1001	73	49	I
0100 1010	74	4A	J
0100 1011	75	4B	K
0100 1100	76	4C	L
0100 1101	77	4D	M
0100 1110	78	4E	N
0100 1111	79	4F	O
0101 0000	80	50	P
0101 0001	81	51	Q
0101 0010	82	52	R
0101 0011	83	53	S
0101 0100	84	54	T
0101 0101	85	55	U
0101 0110	86	56	V
0101 0111	87	57	W
0101 1000	88	58	X
0101 1001	89	59	Y
0101 1010	90	5A	Z
0101 1011	91	5B	[
0101 1100	92	5C	\
0101 1101	93	5D	]
0101 1110	94	5E	^
0101 1111	95	5F	_

Binário	Decimal	Hexa	Glifo
0110 0000	96	60	`
0110 0001	97	61	a
0110 0010	98	62	b
0110 0011	99	63	c
0110 0100	100	64	d
0110 0101	101	65	e
0110 0110	102	66	f
0110 0111	103	67	g
0110 1000	104	68	h
0110 1001	105	69	i
0110 1010	106	6A	j
0110 1011	107	6B	k
0110 1100	108	6C	l
0110 1101	109	6D	m
0110 1110	110	6E	n
0110 1111	111	6F	o
0111 0000	112	70	p
0111 0001	113	71	q
0111 0010	114	72	r
0111 0011	115	73	s
0111 0100	116	74	t
0111 0101	117	75	u
0111 0110	118	76	v
0111 0111	119	77	w
0111 1000	120	78	x
0111 1001	121	79	y
0111 1010	122	7A	z
0111 1011	123	7B	{
0111 1100	124	7C	|
0111 1101	125	7D	}
0111 1110	126	7E	~

Binário	Decimal	Hexa	Abreviatura	Descrição
0000 0000	00	00	NUL	*Null* - Nulo
0000 0001	01	01	SOH	*Start of Header* - Inicio do cabeçalho
0000 0010	02	02	STX	*Start of Text* - Inicio do texto
0000 0011	03	03	ETX	*End of Text* - Fim do texto
0000 0100	04	04	EOT	*End of Tape* - Fim de fita
0000 0101	05	05	ENQ	*Enquire* - Interroga identidade do terminal
0000 0110	06	06	ACK	*Acknowledge* - Reconhecimento
0000 0111	07	07	BEL	*Bell* - Campainha
0000 1000	08	08	BS	*Back-space* - Espaço atrás
0000 1001	09	09	HT	*Horizontal Tabulation* - Tabulação horizontal
0000 1010	10	0A	LF	*Line-Feed* - Alimenta linha
0000 1011	11	0B	VT	*Vertical Tabulation* - Tabulação vertical
0000 1100	12	0C	FF	*Form-Feed* - Alimenta formulário
0000 1101	13	0D	CR	*Carriage-Return* - (enter)
0000 1110	14	0E	SO	*Shift-Out* - Saida do *shift* (passa a usar caracteres de baixo da tecla - minúsculas, etc.)
0000 1111	15	0F	SI	*Shift-In* - Entrada no *shift* (passa a usar caracteres de cima da tecla: maiúsculas, caracteres especiais, etc.)
0001 0000	16	10	DLE	Data-Link Escape
0001 0001	17	11	DC1	Device-Control 1
0001 0010	18	12	DC2	Device-Control 2
0001 0011	19	13	DC3	Device-Control 3
0001 0100	20	14	DC4	Device-Control 4
0001 0101	21	15	NAK	*Neg-Acknowledge* - Não-reconhecimento
0001 0110	22	16	SYN	Synchronous Idle
0001 0111	23	17	ETB	End-of-Transmission Block
0001 1000	24	18	CAN	Cancel
0001 1001	25	19	EM	End-Of-Medium
0001 1010	26	1A	SUB	Substitute
0001 1011	27	1B	ESC	Escape
0001 1100	28	1C	FS	File Separator
0001 1101	29	1D	GS	Group Separator
0001 1110	30	1E	RS	Record Separator
0001 1111	31	1F	US	Unit Separator
0111 1111	127	7F	DEL	Delete

Geração

```
int main () {
char c;
printf ("Codigo\tCaracter\n");
printf ("------\t--------\n");
for (c=32;c<=126;c++)
  printf ("%6d\t%c\n",c,c);
}
```

Strings

Conceitos

- Strings (ou cadeias de caracteres) são seqüências de valores do tipo *char*;
- Não existe, na linguagem C, um tipo primitivo *string*;
- O tipo *string* é implementado como um vetor de caracteres;
- Um string literal é representado por uma cadeia de caracteres delimitada por aspas duplas (como em “ABCD”);
- Internamente, o string é terminado com o caracter ‘\0’, que represeta o número zero;

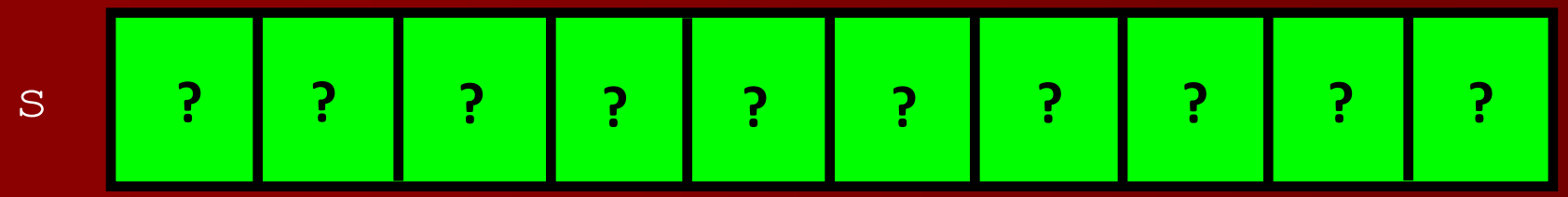
Conceitos

- É um tipo de dados agregado homogêneo em que o tipo do elemento é caracter;
- Serve, portanto, para representar cadeias de caracteres e facilitar a manipulação das mesmas;
- As operações variam muito conforme a linguagem de programação, mas as mais usuais são leitura, escrita, atribuição, concatenação, comparação, pesquisa e segmentação.

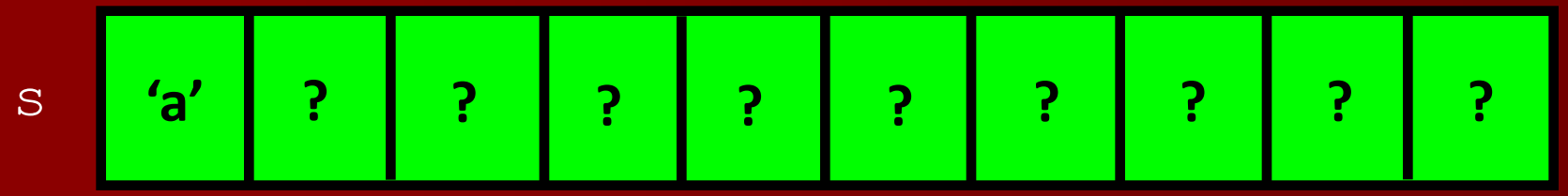
Linguagem C

- Uma cadeia de caracteres é um vetor cujo elemento é do tipo “char”;
- O vetor pode comportar cadeias de caracteres cujo tamanho não exceda o seu próprio tamanho menos 1 (um);
- Toda cadeia de caracteres deve terminar com o caracter especial ‘\0’, que representa o número zero em forma binária e não corresponde a nenhum caracter visível;
- Os elementos podem ser indexados individualmente;

```
char s[10];
```



```
s[0]='a';
```



Linguagem C

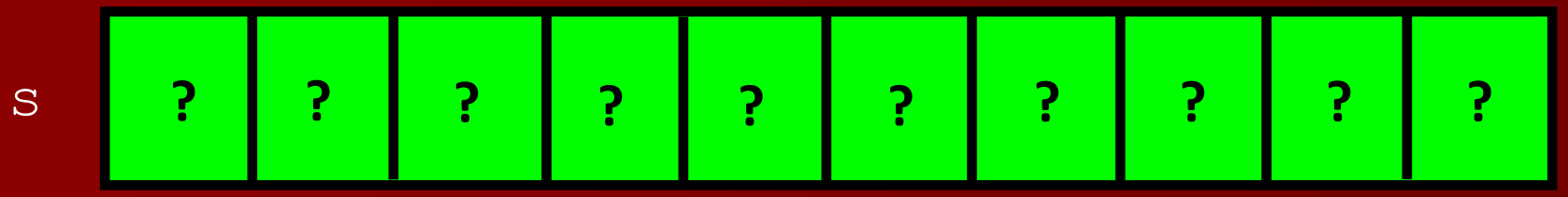
- A biblioteca <stdio.h> disponibiliza as funções:
  - `gets()`
  - `puts()`
- A biblioteca <string.h> disponibiliza uma série de operações para manipulação de strings como um objeto, sem ter que indexar seus elementos individualmente.
- Principais funções:
  - `strlen()`
  - `strchr()`
  - `strstr()`
  - `strcpy()`
  - `strcmp()`
  - `strcat()`

gets

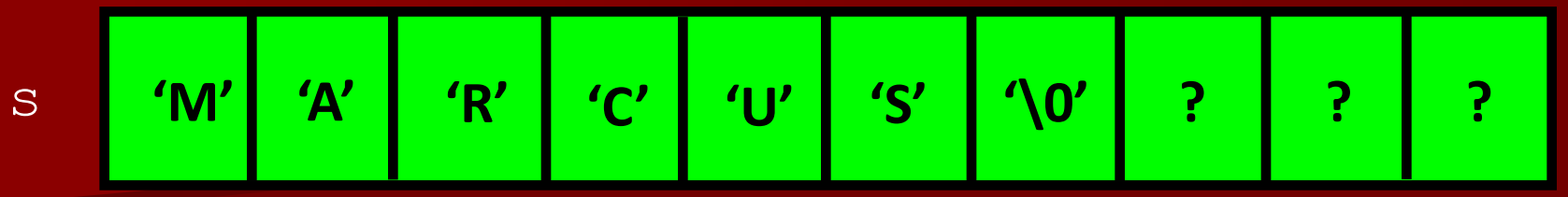
```
gets (char *s)
```

- Aguarda a digitação de uma seqüência de caracteres no teclado pelo usuário, e copia todos eles para o vetor passado como argumento;
- A seqüência deve terminar com ENTER;
- O ENTER não é copiado para o vetor;
- O caracter ‘\0’ é inserido automaticamente depois do último caracter no vetor;
- Não é feita verificação de tamanho máximo na leitura.

```
char s[10];
```




```
gets (s);
```



scanf

```
scanf (“%n[^\n]s”,s)
```

- Alternativa para a função gets ();
- Especifica a quantidade máxima de caracteres que serão copiados da entrada para o vetor;
- Os demais caracteres digitados pelo usuário serão ignorados.
- Exemplo:

```
int main () {
   char s[10];
   scanf ("%5[^\n]s",s);
   printf ("%s\n",s);
   }
```

puts

```
puts (char *s)
```

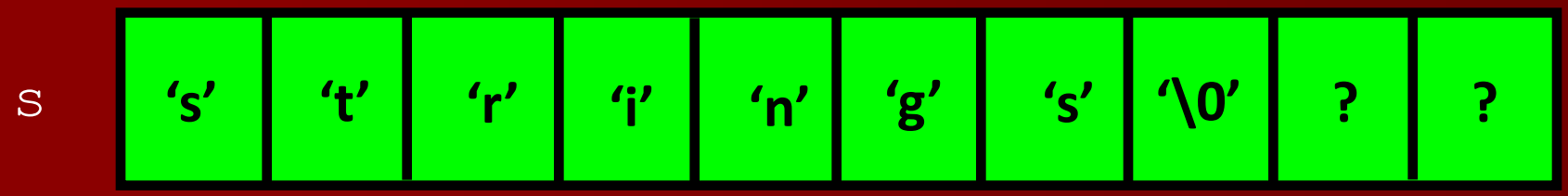
- Envia para o dispositivo de saída padrão (tela) o conteúdo do vetor passado como argumento;
- Pode ser usado como alternativa para
  `printf ("%s",s)`
  que, no entanto, oferece mais opções de formatação.

strcpy

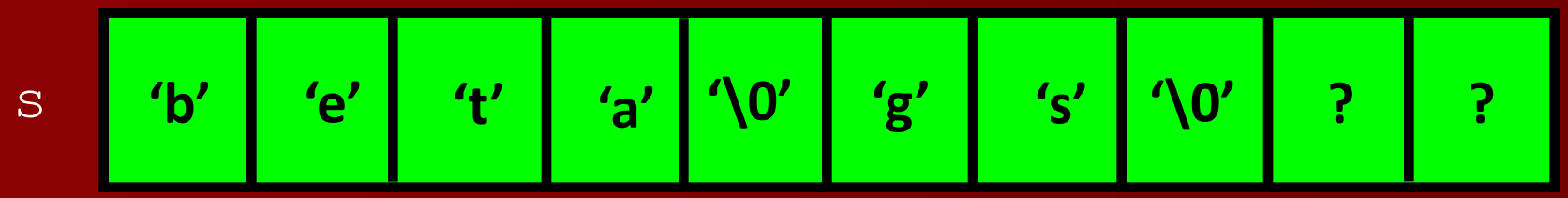
```
char *strcpy (char *s1, char *s2)
```

- Copia os caracteres da cadeia s2 para a cadeia s1 e acrescenta o ‘\0’ no final;
- O conteúdo original de s1 é perdido;
- A cadeia s2 permanece inalterada;
- Retorna um ponteiro para s1.

```
strcpy(s,"strings");
```



```
strcpy(s,"beta");
```

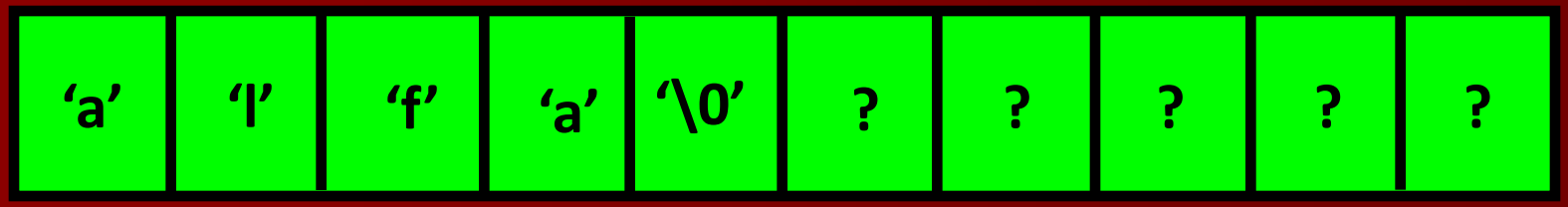


strcat

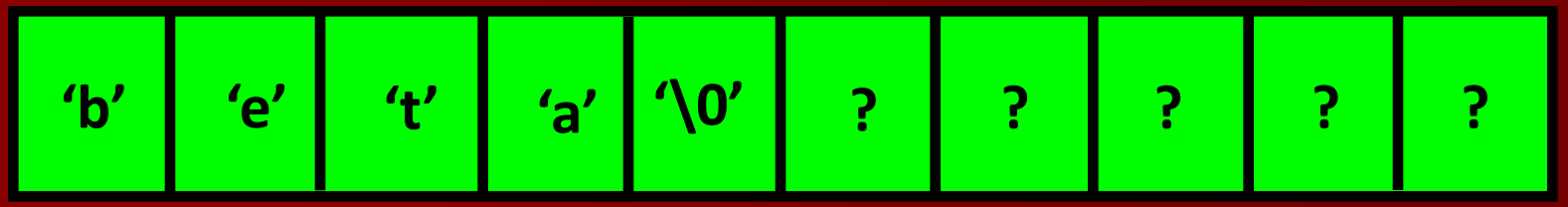
```
char *strcat (char *s1, char *s2)
```

- Copia os caracteres da cadeia s2 para o final da cadeia s1 e acrescenta o ‘\0’ no final;
- A cadeia s2 permanece inalterada;
- Retorna um ponteiro para s1.

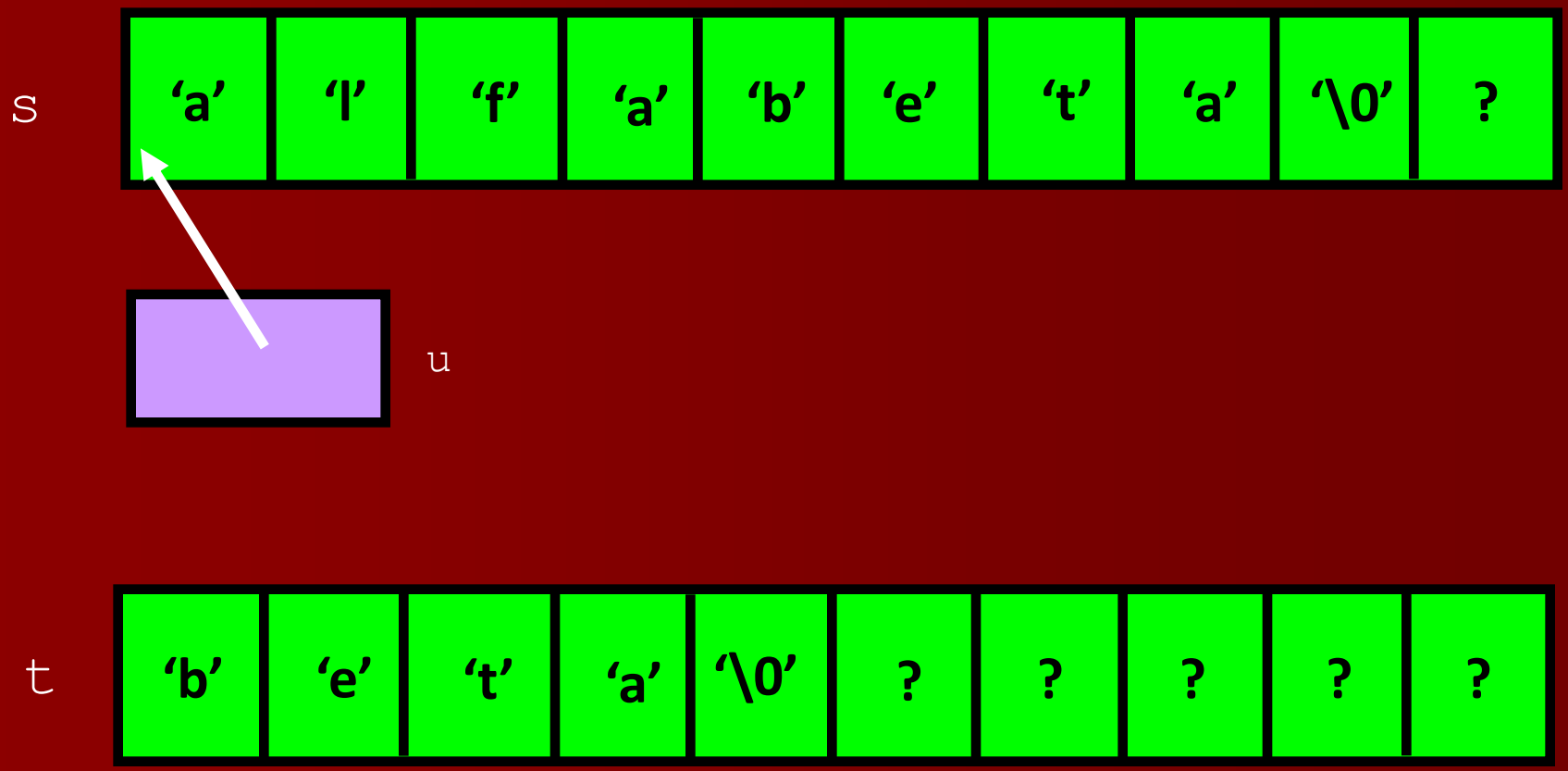
s



t




```
char *u;
u=strcat(s,t);
```

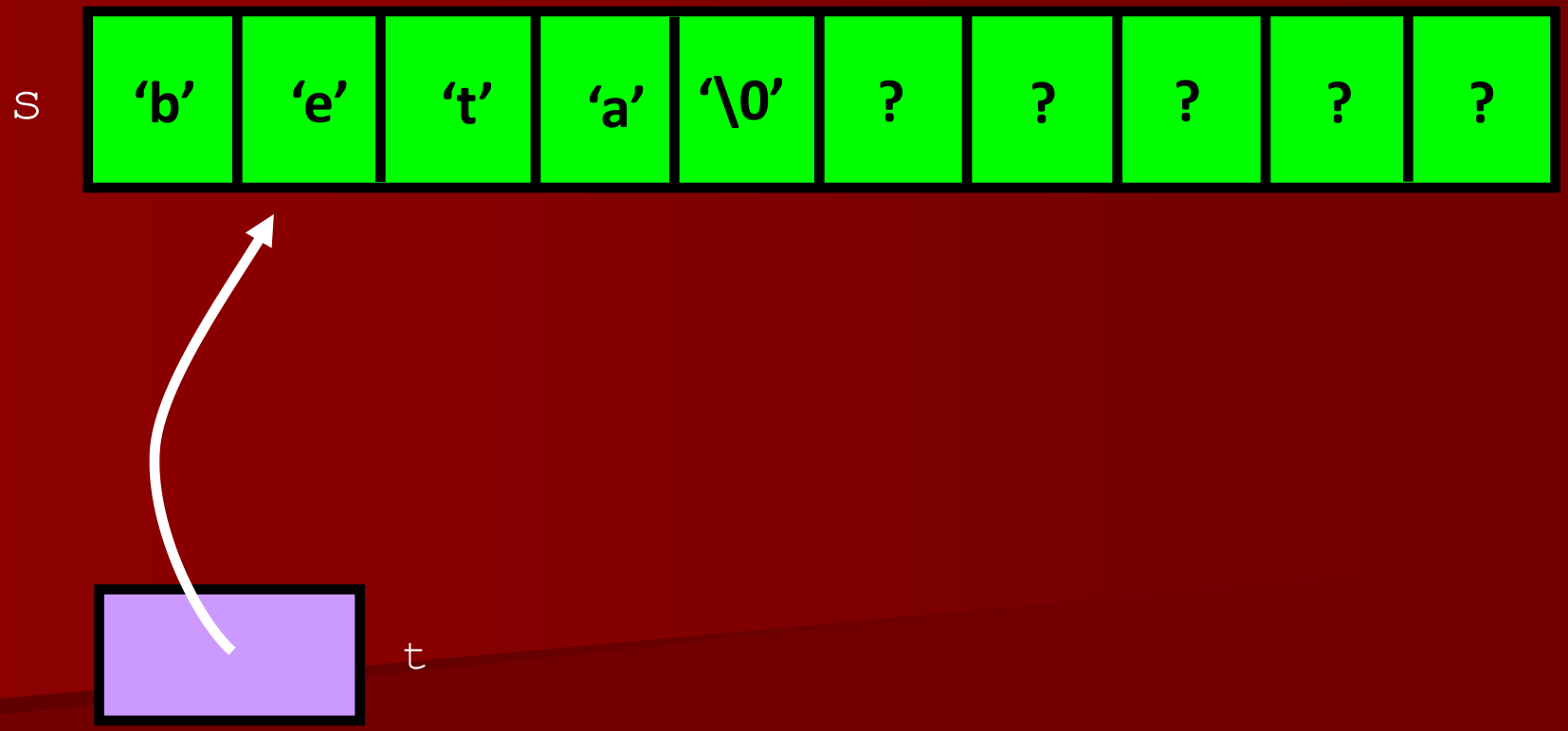

s
'a'
'l'
'f'
'a'
'b'
'e'
't'
'a'
'\0'
?
u
t
'b'
'e'
't'
'a'
'\0'
?
?
?
?
?

strstr

```
char *strstr (char *s1, char *s2)
```

- Procura, dentro da cadeia s1, a subcadeia s2;
- Caso exista, retorna o endereço onde s2 inicia dentro de s1;
- Caso não exista, retorna NULL (endereço zero);
- Se houverem múltiplas ocorrências de s2 em s1, retorna o endereço da primeira ocorrência.

```
char s[10],*t;
strcpy (s,"beta");
t=strstr(s,"et");
```

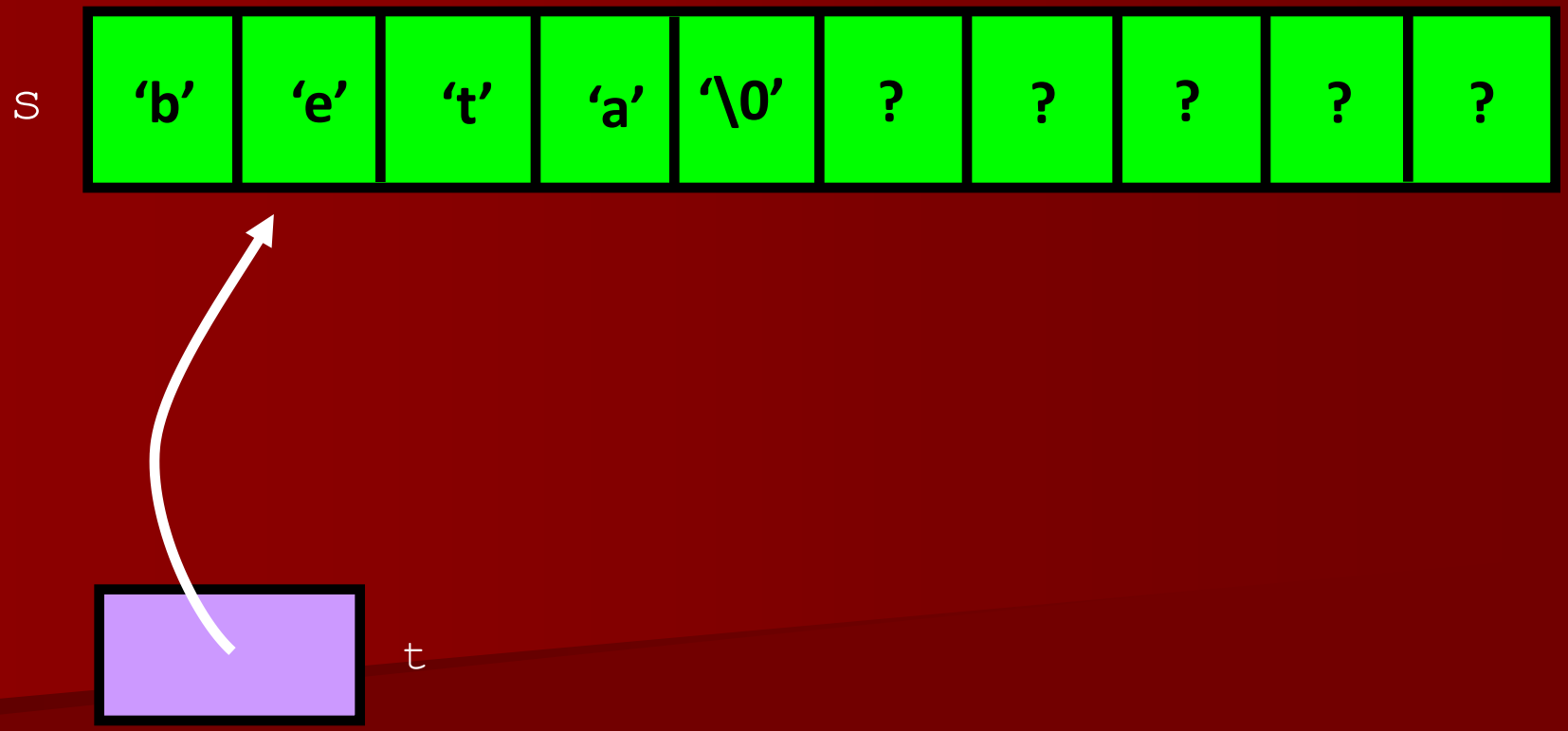


strchr

```
char *strchr (char *s1, char c)
```

- Procura, dentro da cadeia s1, o caracter c;
- Caso exista, retorna o endereço onde c ocorre dentro de s1;
- Caso não exista, retorna NULL (endereço zero);
- Se houverem múltiplas ocorrências de c em s1, retorna o endereço da primeira ocorrência.

```
char s[10],*t;
strcpy (s,"beta");
t=strchr(s,'e');
```



strcmp

```
int strcmp (char *s1, char *s2)
```

- Compara as cadeias s1 e s2 do ponto de vista lexicográfico;
- Retorna:
  - 0 se elas forem idênticas ;
  - um valor qualquer menor que zero se s1 antecede s2;
  - um valor qualquer maior que zero se s2 sucede s1.

Ordenação lexicográfica

```
int strcmp (char *s1, char *s2)
```

- Também conhecida como ordenação alfabética;
- Ordem usada nos dicionários;
- Algoritmo:
  - Compara caracter por caracter, a partir da primeira posição de ambos os argumentos;
  - Enquanto eles forem idênticos, avança para a próxima posição;
  - Se eles forem diferentes numa certa posição, usar a ordem dos caracteres no alfabeto para determinar antecessor/sucessor;
  - Se uma cadeia for prefixo da outra, a que prefixa é considerada a antecessora.

Ordenação lexicográfica

```
int strcmp (char *s1, char *s2)
```

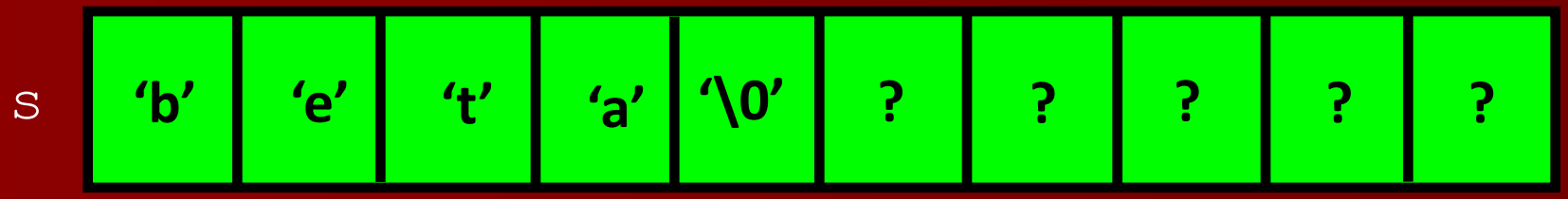
- Exemplos:
  - "ABC" antecede "DEF", portanto `strcmp ("ABC","DEF")<0;`
  - "XYZ" é igual a "XYZ", portanto `strcmp ("XYZ","XYZ") ==0;`
  - "KL" sucede "GDS", portanto `strcmp ("KL","GDS") >0;`
  - "KL" antecede "KML", portanto `strcmp ("KL","KLM") <0;`

strlen

```
int strlen (char *s)
```

- Retorna o comprimento (ou seja, a quantidade de caracteres) da cadeia s;

```
char s[10];
if strlen(s)>=2 ...
```




```
strlen(s)=4
```