(**) 1. Em programas que manipulam strings, a leitura de cada string é feita em um buffer com o tamanho máximo para as strings lidas. Por exemplo, o código abaixo lê uma string de até 1024 caracteres (contando o `'\0'` final):

```
#define BUFLEN 1024
(...)
char buffer [BUFLEN];
fgets (buffer, BUFLEN, stdin);
```

Neste código, o buffer que contém a string possui tamanho 1024, mesmo que a string digitada seja simplesmente “oi”. A maior parte do espaço não é usada. Além disso, se for necessário ler outra string usando o mesmo buffer, o conteúdo da primeira string digitada será substituído. Nestas situações, uma forma de economizar memória é usar o buffer maior apenas para ler as strings, mas alocar cada nova string em outro espaço, que tem apenas o tamanho necessário.

Com base nisso, escreva uma função `char* empacotaString (char* string)`, que recebe como parâmetro um buffer contendo uma string e retorna uma cópia da string, mas em um espaço que tem apenas o tamanho necessário. A nova string deve ser alocada dentro da função, mas a responsabilidade de desalocá-la é do chamador.

(**) 2. Escreva uma função `char* concatenaStrings (char* s1, char* s2)`, que recebe duas strings como parâmetros e cria uma nova string que é uma concatenação das duas. Por exemplo, se `s1` for “foo” e `s2` for “bar”, a função retorna uma string contendo os caracteres “foobar”. A string retornada deve ser alocada dentro da função, e ter apenas o tamanho necessário para conter as duas strings (além do `'\0'` final!). A responsabilidade de desalocar esta string é do chamador.

(☠) 3. Um problema recorrente em aplicações como editores de texto e compiladores é o balanceamento de parênteses: dada uma string, precisamos apontar as correspondências entre os caracteres `'('` e `')'` encontrados. Você já deve ter visto o balanceamento de parênteses em ação ao trabalhar com o Code::Blocks: quando colocamos o cursor sobre um `'('`, o programa destaca o `')'` correspondente, e vice-versa. Escreva uma função com o protótipo a seguir:

```
void balanceamentoDeParenteses (char* str, int* parenteses);
```

A função recebe uma string `str` e um vetor `parenteses` de saída. Você pode pressupor que a string é bem-formada (ou seja, que é terminada em `'\0'`) e que o vetor de saída tem uma posição para cada caractere da string, exceto o `'\0'`. A função deve preencher o vetor de saída da seguinte forma:

1) Se o caractere em uma posição `i` da string é um `'('`, a posição `i` do vetor deve dizer a posição da string onde está o `')'` correspondente, e vice-versa.

2) Se o caractere em uma posição `i` da string é um `'('` sem um `')'` correspondente ou vice-versa, a posição `i` do vetor deve conter o valor -1.

3) Se o caractere em uma posição `i` da string não é um `'('` nem um `')'`, a posição `i` do vetor deve conter o valor 0.

Exemplos (linha 1: posição, linha 2: string, linha 3: vetor de saída):

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
i	f	(	(	a	&	&	b	)	|	|	(	c	&	&	d	)	)	{	\0
0	0	17	8	0	0	0	0	3	0	0	16	0	0	0	0	11	2	0	

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
i	f	(	(	a	&	&	b	)	)	|	|	(	c	&	&	d	)	)	\0
0	0	9	8	0	0	0	0	3	2	0	0	17	0	0	0	0	12	-1	

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
{	[	(	(	(	)	(	)	)	(	)	]	}	(	(	)	(	(	)	\0
0	0	-1	8	5	4	7	6	3	10	9	0	0	-1	15	14	-1	18	17	

(****) 4. (Opcional!) Escreva um programa que lê 3 strings digitadas pelo usuário. O programa deve imprimir as strings em seguida, uma abaixo da outra. O usuário pode então digitar uma nova string. A cada string nova digitada, o programa deve "rolar" as strings, descartando a mais antiga. Uma restrição é que você não pode copiar o conteúdo de um buffer para outro. O programa deve terminar quando o usuário digitar uma string sem caracteres.

Para simular o efeito do rolamento, apague todo o conteúdo da tela e mostre novamente as strings, na ordem apropriada. Para limpar a tela, use a função `system` da stdlib, passando como parâmetro a string `"cls"` ou `"clear"` (se você está usando Windows ou Linux, respectivamente – para computadores Apple, será preciso pesquisar). Exemplo de saída:

(no início)
```
string1
string2
string3
```

(usuário digita `mais uma`):
```
string2
string3
mais uma
```

(usuário digita `outra`):
```
string3
mais uma
outra
```

(usuário digita `foobar`):
```
mais uma
outra
foobar
```