Programação Estruturada
Prof. Rodrigo Hausen
http://progest.compscinet.org

Exercícios com ponteiros

EXERCÍCIOS COM PONTEIROS

Última aula de laboratório:

```
void troca(int *x, int *y) {
  int aux = *x;
  *x = *y;
  *y = aux;
}
```

```
[cling]$ int x = 42, y = 18;
[cling]$ troca(&x, &y);
[cling]$ x
(int) 18
[cling]$ y
(int) 42
```

ORDENAÇÃO POR BORBULHAMENTO (bubble sort)

Entrada: array A de int
Resultado: ordena A em ordem crescente

- enquanto houver par de elementos A[i], A[i+1] tal que A[i] > A[i+1] (onde i = 0…n-2)
  - troca A[i] com A[i+1]

Ordenação **in-place**: o próprio array é ordenado, sem criação de outro array auxiliar.

ORDENAÇÃO POR BORBULHAMENTO (bubble sort)

Implemente no arquivo ordena.c:

- **void** bubblesort(**int** *a, **int** n)

Use a função troca.

Teste no cling:

```
[cling]$ .L ordena.c
[cling]$ int v[] = { 42, 18, 99, -1, 3 };
[cling]$ bubblesort(v, 5);
[cling]$ v
```

ORDENAÇÃO POR BORBULHAMENTO (bubble sort)

Implemente no arquivo ordena.c:

- **void** bubblesort(**int** *a, **int** n)

Use a função troca.

Teste no cling:

```
[cling]$ .L ordena.c
[cling]$ int v[] = { 42, 18, 99, -1, 3 };
[cling]$ bubblesort(v, 5);
[cling]$ v
(int [5]) { -1, 3, 18, 42, 99 }
[cling]$
```

BREVE INTRODUÇÃO À BIBLIOTECA string.h

- Relembrando: a linguagem C não possui string como tipo primitivo. Para tal, usamos ponteiros para char ou vetores de char terminados pelo caractere nulo '\0'

- Todas as operações com “strings” devem ser feitas por meio de funções.

- Em C, a biblioteca padrão para manipulação de “strings” é a string.h

BREVE INTRODUÇÃO À BIBLIOTECA string.h

- Para usar as funções de string da biblioteca padrão, devemos incluir o arquivo string.h no início do nosso programa

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

- Assim, podemos usar as funções de strings definidas nesta biblioteca: determinar tamanho, duplicar, concatenar, comparar, procurar, etc.

BREVE INTRODUÇÃO À BIBLIOTECA string.h

- Determinar tamanho de string:

  size_t strlen(**const char** ***s**)

  A função strlen retorna a quantidade de bytes ocupados pela string s, sem incluir o caractere nulo final.

  Exemplo no cling:

```
[cling]$ #include <string.h>
[cling]$ strlen("bom dia!")
```

BREVE INTRODUÇÃO À BIBLIOTECA string.h

- Determinar tamanho de string:

  size_t strlen(**const char** ***s**)

  A função strlen retorna a quantidade de bytes ocupados pela string s, sem incluir o caractere nulo final.

  Exemplo no cling:

```
[cling]$ #include <string.h>
[cling]$ strlen("bom dia!")
(size_t) 8
[cling]$
```

BREVE INTRODUÇÃO À BIBLIOTECA string.h

- Determinar tamanho de string:

  size_t strlen(**const char** ***s**)

Obs1: O tipo **size_t** é um tipo próprio para descrever tamanhos em bytes.

Obs2: O tamanho em bytes não é a mesma coisa que número de caracteres.

p. ex. na máquina virtual, strlen("água") é 5 pois a forma que ela usa para representar “á” requer 2 bytes para essa letra.

BREVE INTRODUÇÃO À BIBLIOTECA string.h

- Duplicar string:

  char ***strdup(**const char** ***s)**

  Retorna um ponteiro para uma cópia da string original.

  Para que serve? Mesmo que a string original seja imutável, a cópia pode ser alterada!

  Atenção: após o uso, devemos dar free no ponteiro!

BREVE INTRODUÇÃO À BIBLIOTECA string.h

- **char** ***strdup(const char** ***s)**

```
[cling]$ char *s = "ana";
```

BREVE INTRODUÇÃO À BIBLIOTECA string.h

- **char** ***strdup(const char** ***s)**

```
[cling]$ char *s = "ana";
```

Aviso: tentando atribuir constante a uma região mutável.

```
[cling]$
```

BREVE INTRODUÇÃO À BIBLIOTECA string.h

- **char** ***strdup(const char** ***s)**

```
[cling]$ char *s = "ana";
```

Aviso: tentando atribuir constante a uma região mutável.

```
[cling]$ s
```

BREVE INTRODUÇÃO À BIBLIOTECA string.h

- **char** ***strdup(const char** ***s**)

```
[cling]$ char *s = "ana";
```

Aviso: tentando atribuir constante a uma região mutável.

```
[cling]$ s
(char *) "ana"
[cling]$
```

BREVE INTRODUÇÃO À BIBLIOTECA string.h

- **char** ***strdup(const char** ***s**)

```
[cling]$ char *s = "ana";
```
Aviso: tentando atribuir constante a uma região mutável.
```
[cling]$ s
(char *) "ana"
[cling]$ s[0] = 'A'
```

BREVE INTRODUÇÃO À BIBLIOTECA string.h

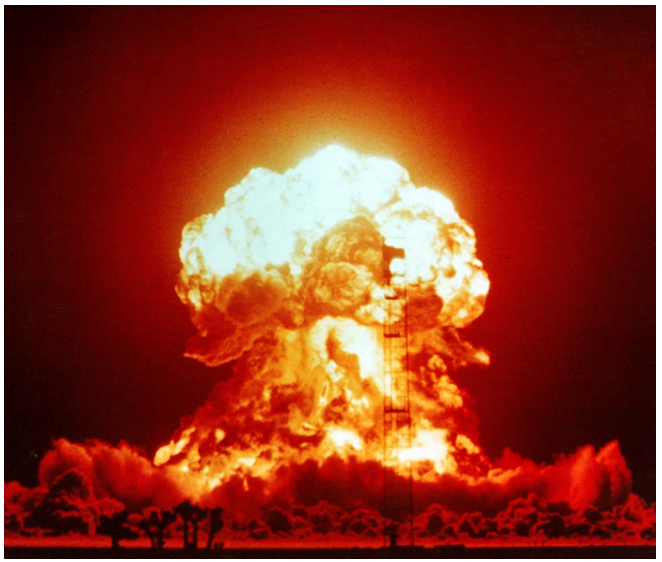
- **char** *strdup(**const char** *s)

```
[cling]$ char *s = "ana";
```

Aviso: tentando atribuir constante a uma região mutável.

```
[cling]$ s
(char *) "ana"
[cling]$ s[0] = 'A'
```

Falha de segmentação

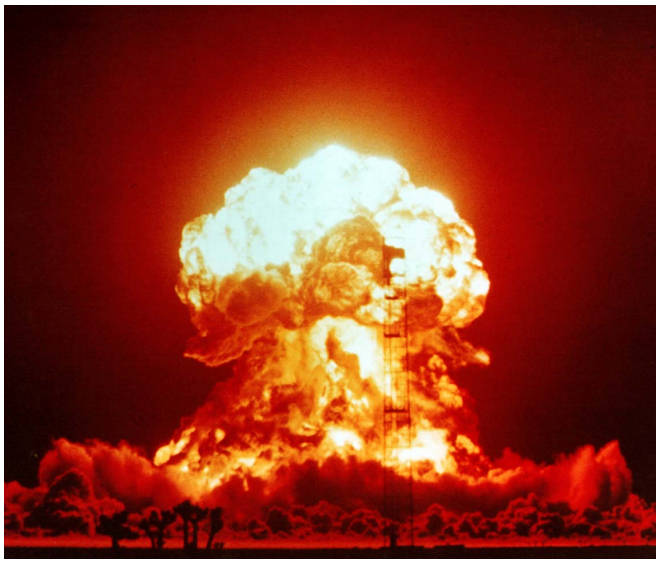


BREVE INTRODUÇÃO À BIBLIOTECA string.h

- **char** *strdup(**const char** *s)

```
[cling]$ char *s = "ana";
```
Aviso: tentando atribuir constante a uma região mutável.
```
[cling]$ s
(char *) "ana"
[cling]$ s[0] = 'A'
```
Falha de segmentação

Tentei alterar região imutável



BREVE INTRODUÇÃO À BIBLIOTECA string.h

- **char** ***strdup(const char** ***s)**

```
[cling]$ #include <string.h>
[cling]$
```

BREVE INTRODUÇÃO À BIBLIOTECA string.h

- **char** ***strdup(const char** ***s)**

```
[cling]$ #include <string.h>
[cling]$ const char *s = "ana";
```

BREVE INTRODUÇÃO À BIBLIOTECA string.h

- **char** ***strdup(const char** ***s**)

```
[cling]$ #include <string.h>
[cling]$ const char *s = "ana";
[cling]$
```

BREVE INTRODUÇÃO À BIBLIOTECA string.h

- **char** ***strdup(const char** ***s**)

```
[cling]$ #include <string.h>
[cling]$ const char *s = "ana";
[cling]$ char *p = strdup(s);
```

BREVE INTRODUÇÃO À BIBLIOTECA string.h

- **char** ***strdup(const char** ***s**)

```
[cling]$ #include <string.h>
[cling]$ const char *s = "ana";
[cling]$ char *p = strdup(s);
[cling]$
```

BREVE INTRODUÇÃO À BIBLIOTECA string.h

- **char** ***strdup(const char** ***s**)

```
[cling]$ #include <string.h>
[cling]$ const char *s = "ana";
[cling]$ char *p = strdup(s);
[cling]$ p
```

BREVE INTRODUÇÃO À BIBLIOTECA string.h

- **char** ***strdup(const char** ***s**)

```
[cling]$ #include <string.h>
[cling]$ const char *s = "ana";
[cling]$ char *p = strdup(s);
[cling]$ p
(char *) "ana"
[cling]$
```

BREVE INTRODUÇÃO À BIBLIOTECA string.h

- **char** ***strdup(const char** ***s**)

```
[cling]$ #include <string.h>
[cling]$ const char *s = "ana";
[cling]$ char *p = strdup(s);
[cling]$ p
(char *) "ana"
[cling]$ p[0] = 'A';
```

BREVE INTRODUÇÃO À BIBLIOTECA string.h

- **char** ***strdup(const char** ***s**)

```
[cling]$ #include <string.h>
[cling]$ const char *s = "ana";
[cling]$ char *p = strdup(s);
[cling]$ p
(char *) "ana"
[cling]$ p[0] = 'A';
[cling]$
```

BREVE INTRODUÇÃO À BIBLIOTECA string.h

- **char** ***strdup(const char** ***s**)

```
[cling]$ #include <string.h>
[cling]$ const char *s = "ana";
[cling]$ char *p = strdup(s);
[cling]$ p
(char *) "ana"
[cling]$ p[0] = 'A';
[cling]$ p
```

BREVE INTRODUÇÃO À BIBLIOTECA string.h

- **char** ***strdup(const char** ***s**)

```
[cling]$ #include <string.h>
[cling]$ const char *s = "ana";
[cling]$ char *p = strdup(s);
[cling]$ p
(char *) "ana"
[cling]$ p[0] = 'A';
[cling]$ p
(char *) "Ana"
[cling]$
```

BREVE INTRODUÇÃO À BIBLIOTECA string.h

- **char** ***strdup(const char** ***s**)

```
[cling]$ #include <string.h>
[cling]$ const char *s = "ana";
[cling]$ char *p = strdup(s);
[cling]$ p
(char *) "ana"
[cling]$ p[0] = 'A';
[cling]$ p
(char *) "Ana"
[cling]$ free(p);
```

BREVE INTRODUÇÃO À BIBLIOTECA string.h

- **char** *strdup(**const char** *s)

```
[cling]$ #include <string.h>
[cling]$ const char *s = "ana";
[cling]$ char *p = strdup(s);
[cling]$ p
(char *) "ana"
[cling]$ p[0] = 'A';
[cling]$ p
(char *) "Ana"
[cling]$ free(p);
[cling]$
```

BREVE INTRODUÇÃO À BIBLIOTECA string.h

- Copiar string para outra:

  strncpy(**char** *dest, **const char** *src,
  size_t n)

  Copia, no máximo, n bytes de src para dest, incluindo o caractere nulo '\0'.

  CUIDADO: se o caractere nulo não estiver entre os primeiros n bytes de src, ele **não** será copiado automaticamente!

BREVE INTRODUÇÃO À BIBLIOTECA string.h

- strncpy(**char** ***dest**, **const char** ***src**,
  size_t n)

```
[cling]$ #include <string.h>
[cling]$ const char *s;
[cling]$
```

BREVE INTRODUÇÃO À BIBLIOTECA string.h

- strncpy(**char** ***dest**, **const char** ***src**,
  size_t n)

```
[cling]$ #include <string.h>
[cling]$ const char *s;
[cling]$ s = "Corinthians vai ganhar!";
```

BREVE INTRODUÇÃO À BIBLIOTECA string.h

- strncpy(**char** ***dest**, **const char** ***src**,
  size_t n)

```
[cling]$ #include <string.h>
[cling]$ const char *s;
[cling]$ s = "Corinthians vai ganhar!";
[cling]$
```

BREVE INTRODUÇÃO À BIBLIOTECA string.h

- strncpy(**char** ***dest**, **const char** ***src**,
  size_t n)

```
[cling]$ #include <string.h>
[cling]$ const char *s;
[cling]$ s = "Corinthians vai ganhar!";
[cling]$ char *p = strdup(s);
```

BREVE INTRODUÇÃO À BIBLIOTECA string.h

- strncpy(**char** ***dest**, **const char** ***src**,
  size_t n)

```
[cling]$ #include <string.h>
[cling]$ const char *s;
[cling]$ s = "Corinthians vai ganhar!";
[cling]$ char *p = strdup(s);
[cling]$
```

BREVE INTRODUÇÃO À BIBLIOTECA string.h

- strncpy(**char** ***dest**, **const char** ***src**,
  size_t n)

```
[cling]$ #include <string.h>
[cling]$ const char *s;
[cling]$ s = "Corinthians vai ganhar!";
[cling]$ char *p = strdup(s);
[cling]$ p
```

BREVE INTRODUÇÃO À BIBLIOTECA string.h

- strncpy(**char** ***dest**, **const char** ***src**,
  size_t n)

```
[cling]$ #include <string.h>
[cling]$ const char *s;
[cling]$ s = "Corinthians vai ganhar!";
[cling]$ char *p = strdup(s);
[cling]$ p
(char *) "Corinthians vai ganhar!"
[cling]$
```

BREVE INTRODUÇÃO À BIBLIOTECA string.h

- strncpy(**char** ***dest**, **const char** ***src**,
  size_t n)

```
[cling]$ #include <string.h>
[cling]$ const char *s;
[cling]$ s = "Corinthians vai ganhar!";
[cling]$ char *p = strdup(s);
[cling]$ p
(char *) "Corinthians vai ganhar!"
[cling]$ strlen("Figueirense")
```

BREVE INTRODUÇÃO À BIBLIOTECA string.h

- strncpy(**char** ***dest**, **const char** ***src**,
  size_t n)

```
[cling]$ #include <string.h>
[cling]$ const char *s;
[cling]$ s = "Corinthians vai ganhar!";
[cling]$ char *p = strdup(s);
[cling]$ p
(char *) "Corinthians vai ganhar!"
[cling]$ strlen("Figueirense")
(size_t) 11
[cling]$
```

BREVE INTRODUÇÃO À BIBLIOTECA string.h

- strncpy(**char** ***dest**, **const char** ***src**,
  size_t n)

```
[cling]$ #include <string.h>
[cling]$ const char *s;
[cling]$ s = "Corinthians vai ganhar!";
[cling]$ char *p = strdup(s);
[cling]$ p
(char *) "Corinthians vai ganhar!"
[cling]$ strlen("Figueirense")
(size_t) 11
[cling]$ strncpy(p, "Figueirense", 11);
```

BREVE INTRODUÇÃO À BIBLIOTECA string.h

- strncpy(**char** *dest, **const char** *src,
  size_t n)

```
[cling]$ #include <string.h>
[cling]$ const char *s;
[cling]$ s = "Corinthians vai ganhar!";
[cling]$ char *p = strdup(s);
[cling]$ p
(char *) "Corinthians vai ganhar!"
[cling]$ strlen("Figueirense")
(size_t) 11
[cling]$ strncpy(p, "Figueirense", 11);
[cling]$ p
```

BREVE INTRODUÇÃO À BIBLIOTECA string.h

- strncpy(**char** *dest, **const char** *src, **size_t** n)

```
[cling]$ #include <string.h>
[cling]$ const char *s;
[cling]$ s = "Corinthians vai ganhar!";
[cling]$ char *p = strdup(s);
[cling]$ p
(char *) "Corinthians vai ganhar!"
[cling]$ strlen("Figueirense")
(size_t) 11
[cling]$ strncpy(p, "Figueirense", 11);
[cling]$ p
(char *) "Figueirense vai ganhar!"
[cling]$
```

BREVE INTRODUÇÃO À BIBLIOTECA string.h

- strncpy(**char** *dest, **const char** *src, **size_t** n)

```
[cling]$ #include <string.h>
[cling]$ const char *s;
[cling]$ s = "Corinthians vai ganhar!";
[cling]$ char *p = strdup(s);
[cling]$ p
(char *) "Corinthians vai ganhar!"
[cling]$ strlen("Figueirense")
(size_t) 11
[cling]$ strncpy(p, "Figueirense", 11);
[cling]$ p
(char *) "Figueirense vai ganhar!"
[cling]$ free(p)
```

USANDO A BIBLIOTECA string.h

- Crie o arquivo mystr.c
- Implemente a função abaixo para alocar espaço e retornar a concatenação das duas strings:

```
char *mystr_cat(const char *s1,
                const char *s2)
```

- Teste no cling

```
mystr_cat("UF", "ABC") retorna "UFABC"
mystr_cat("UF", "") retorna "UF"
mystr_cat("", "") retorna ""
```

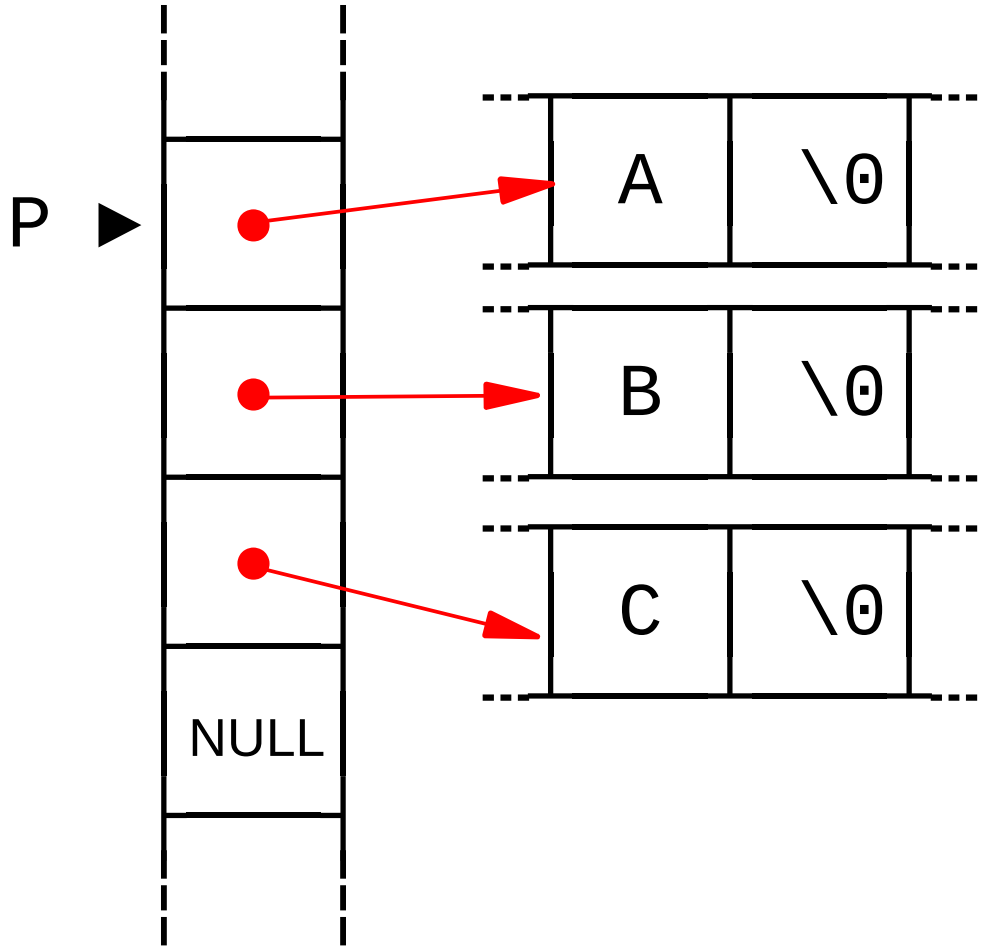
USANDO A BIBLIOTECA string.h

- Implemente a função abaixo para separar uma string em um array de strings, cada uma contendo um caractere da string original. O array deve ser terminado com NULL para indicar o seu fim

```
char **mystr_explode(const char *s)
```

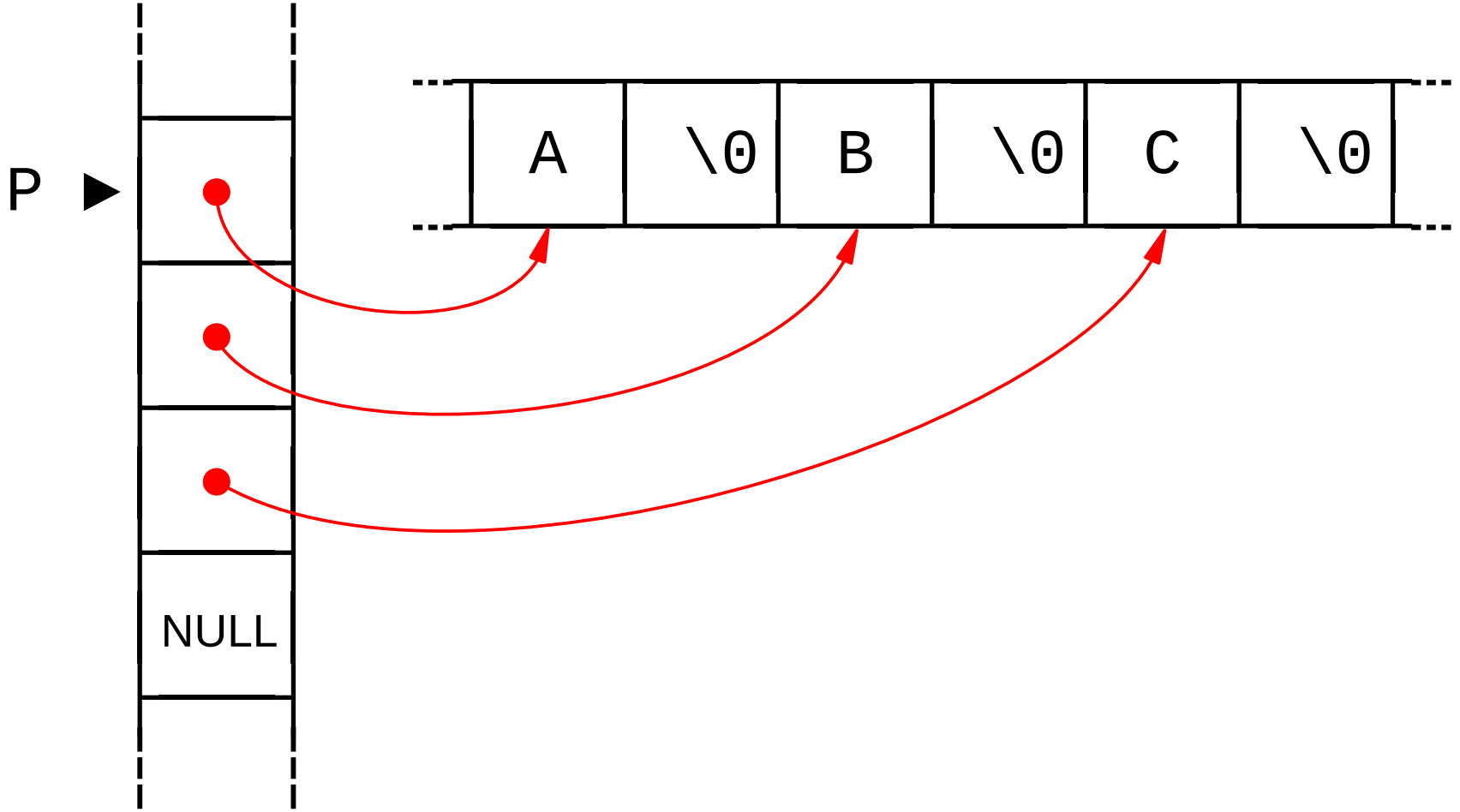
- Implemente também a função mystr_explode_free para liberar todas as regiões de memória alocadas por mystr_explode.

```
[cling]$ .L mystr.c
[cling]$ char **p = mystr_explode("ABC");
[cling]$ p[0]
(char *) "A"
[cling]$ p[1]
(char *) "B"
[cling]$ p[2]
(char *) "C"
[cling]$ p[3]
(char *) nullptr
```

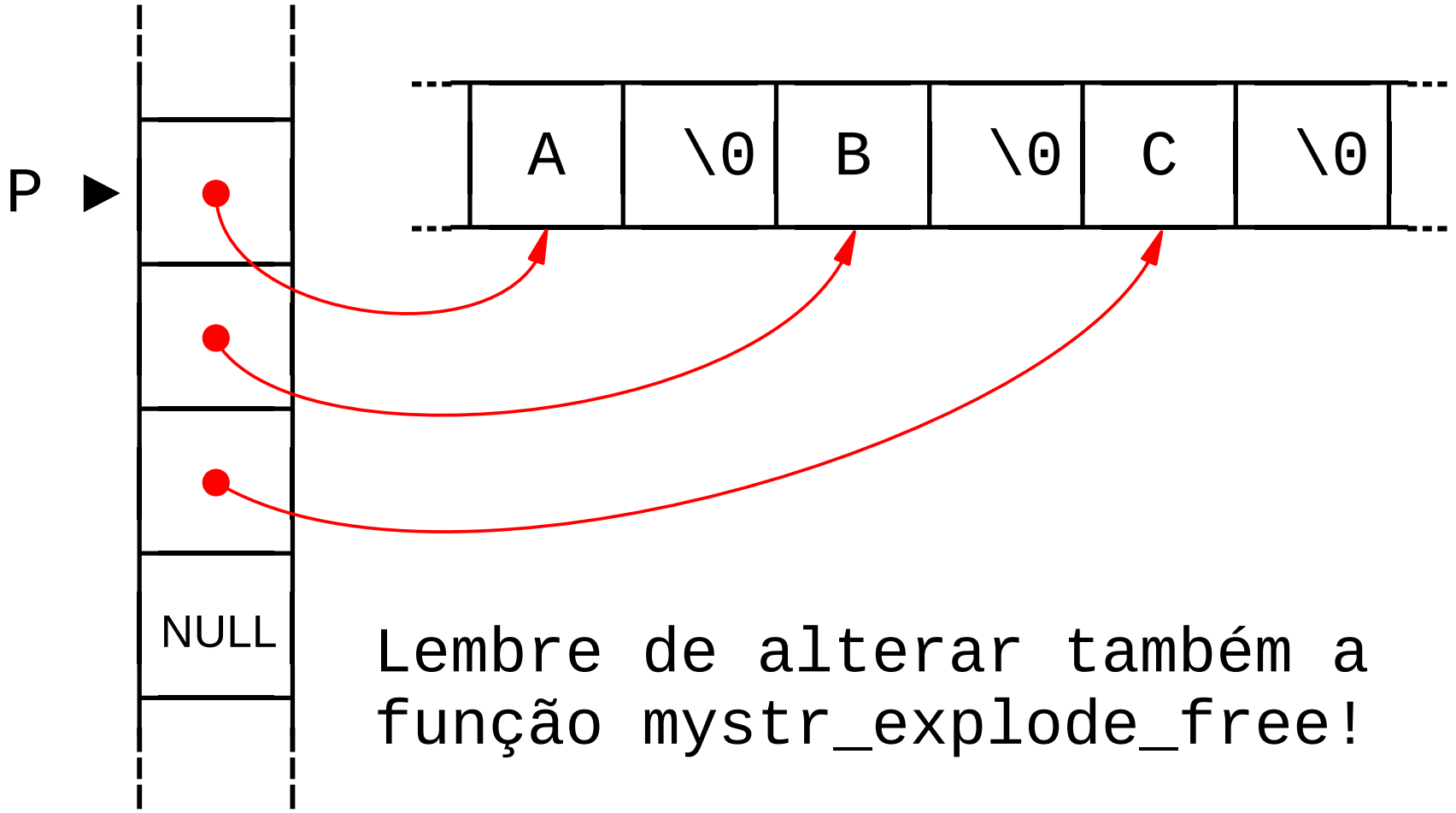


```
[cling]$ .L mystr.c
[cling]$ char **p = mystr_explode("ABC");
[cling]$ p[0]
(char *) "A"
[cling]$ p[1]
(char *) "B"
[cling]$ p[2]
(char *) "C"
[cling]$ p[3]
(char *) nullptr
[cling]$ mystr_explode_free(p);
[cling]$ p = mystr_explode("");
[cling]$ p[0]
(char *) nullptr
[cling]$ mystr_explode_free(p);
```

Altere a função mystr_explode para que, em vez de alocar N regiões de tamanho 2 para as strings, ela aloque apenas uma região contígua de tamanho 2N.



Altere a função mystr_explode para que, em vez de alocar N regiões de tamanho 2 para as strings, ela aloque apenas uma região contígua de tamanho 2N.



Lembre de alterar também a função mystr_explode_free!

BREVE INTRODUÇÃO À BIBLIOTECA string.h

- Procurar uma string em outra (“agulha no palheiro”):

  const char *strstr(**const char** **palheiro*,
  const char **agulha*)

  Retorna um ponteiro para a primeira ocorrência da string *agulha* dentro de *palheiro*.

```
[cling]$ #include <string.h>
[cling]$ strstr("cabeleireiro", "ei")
(const char *) "eireiro"
```

USANDO A BIBLIOTECA string.h

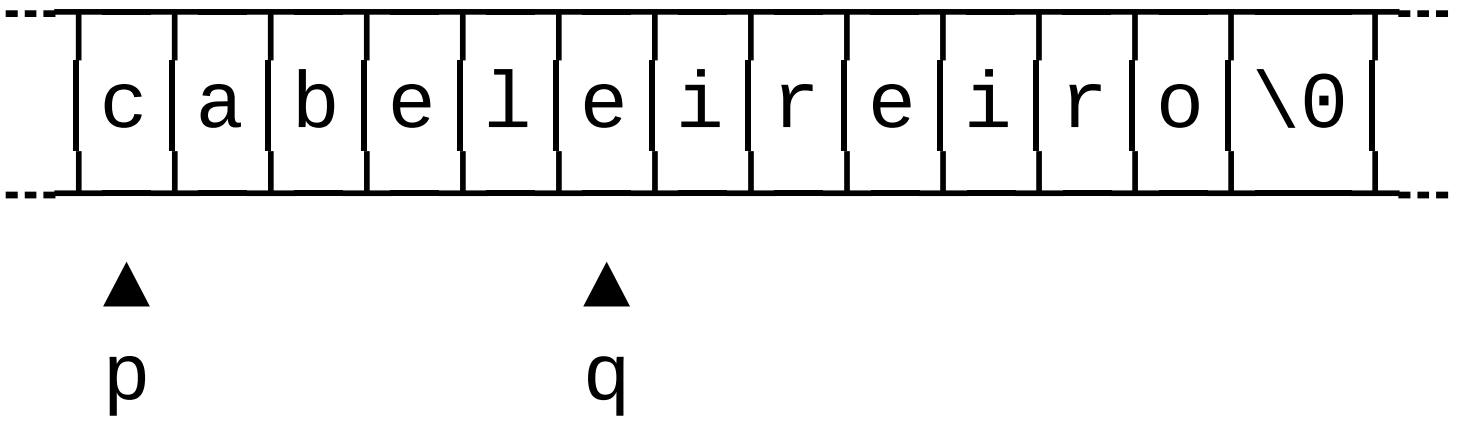
- Implemente a função abaixo, que retorna o número de ocorrências, sem considerar sobreposições, da string agulha na string palheiro

```
int mystr_ocorrencias(
            const char *palheiro,
            const char *agulha)
```

```
[cling]$ .L mystr.c
[cling]$ mystr_ocorrencias("cabeleireiro",
                           "ei")
(int) 2
[cling]$ mystr_ocorrencias("zzzzzzz", "zz")
(int) 3
```

CONTANDO OCORRENCIAS

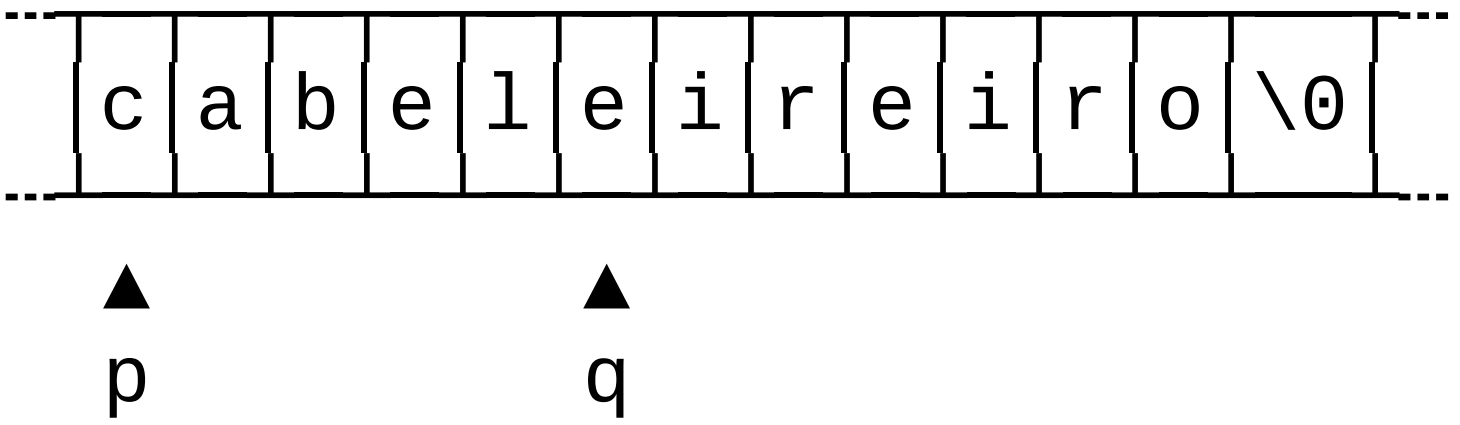
mystr_ocorrencias(p, a)
p é "cabeleireiro", a é "ei"



q = strstr(p, a)

CONTANDO OCORRENCIAS

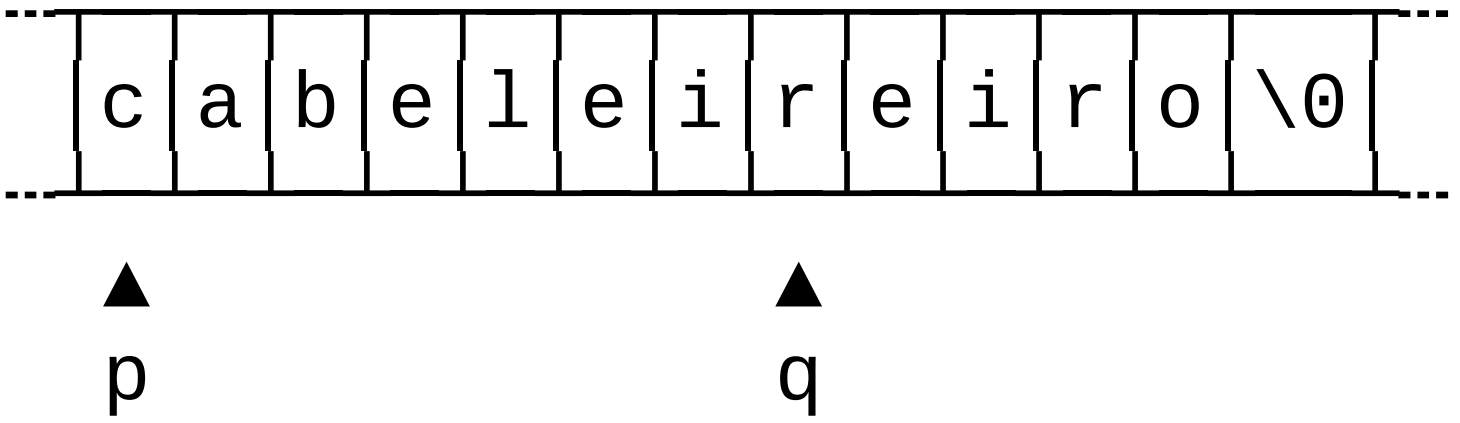
```
mystr_ocorrencias(p, a)
p é "cabeleireiro", a é "ei"
```




```
q = strstr(p, a)
q += strlen(a)
```

CONTANDO OCORRENCIAS

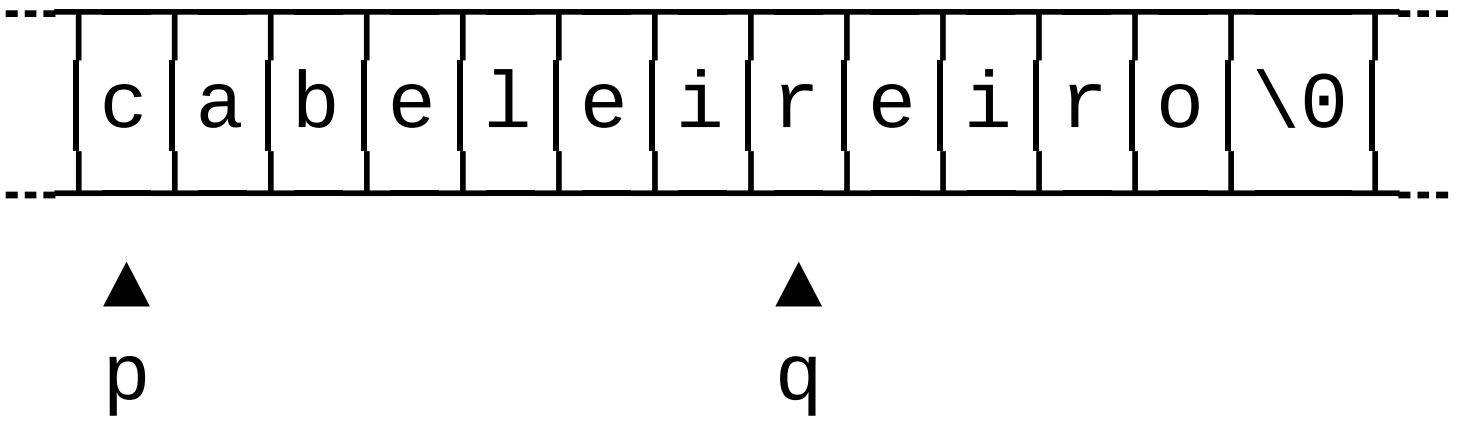
mystr_ocorrencias(p, a)
p é "cabeleireiro", a é "ei"




```
q = strstr(p, a)
q += strlen(a)
```

CONTANDO OCORRENCIAS

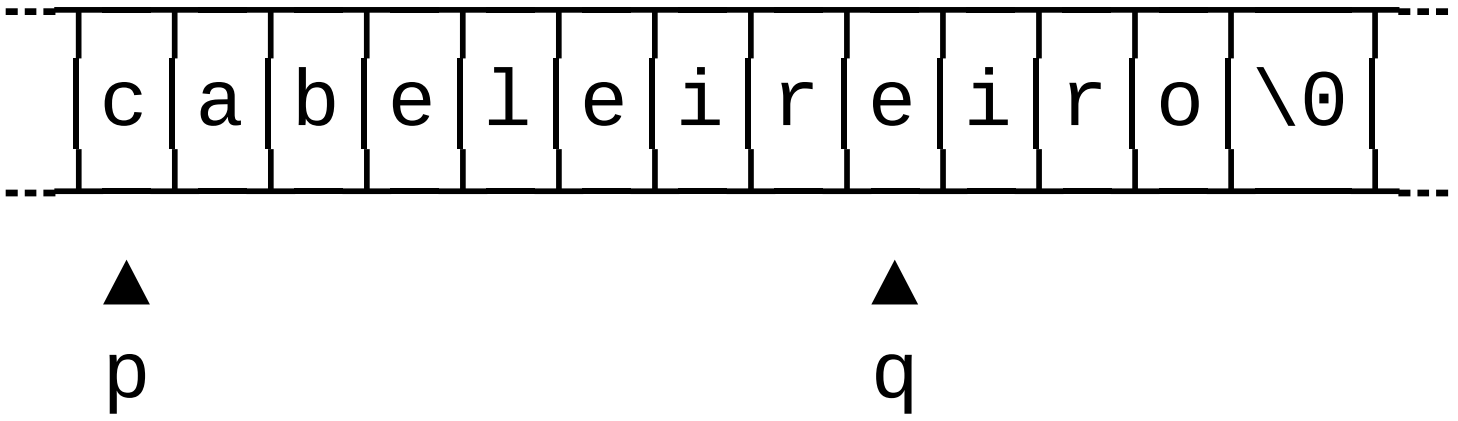
mystr_ocorrencias(p, a)
p é "cabeleireiro", a é "ei"




```
q = strstr(p, a)
q += strlen(a)
q = strstr(q, a)
```

CONTANDO OCORRENCIAS

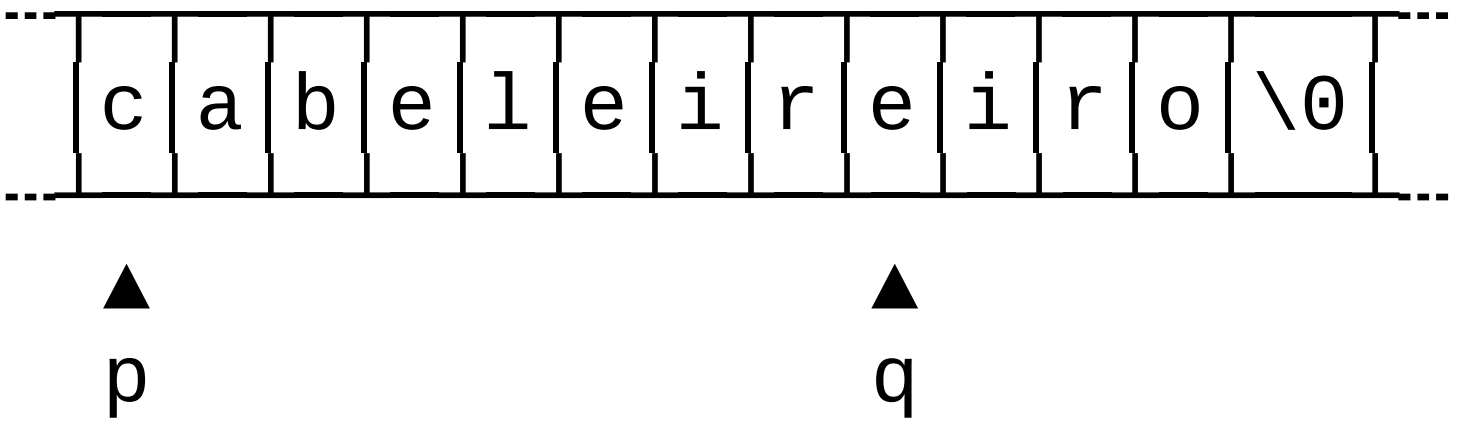
mystr_ocorrencias(p, a)
p é "cabeleireiro", a é "ei"




```
q = strstr(p, a)
q += strlen(a)
q = strstr(q, a)
```

CONTANDO OCORRENCIAS

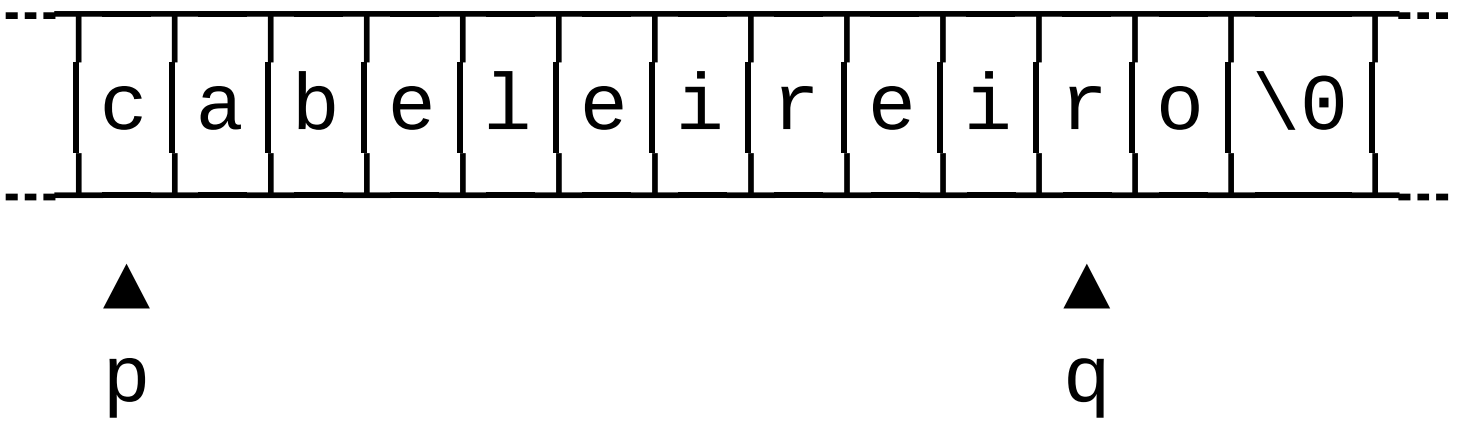
```
mystr_ocorrencias(p, a)
p é "cabeleireiro", a é "ei"
```




```
q = strstr(p, a)
q += strlen(a)
q = strstr(q, a)
q += strlen(a)
```

CONTANDO OCORRENCIAS

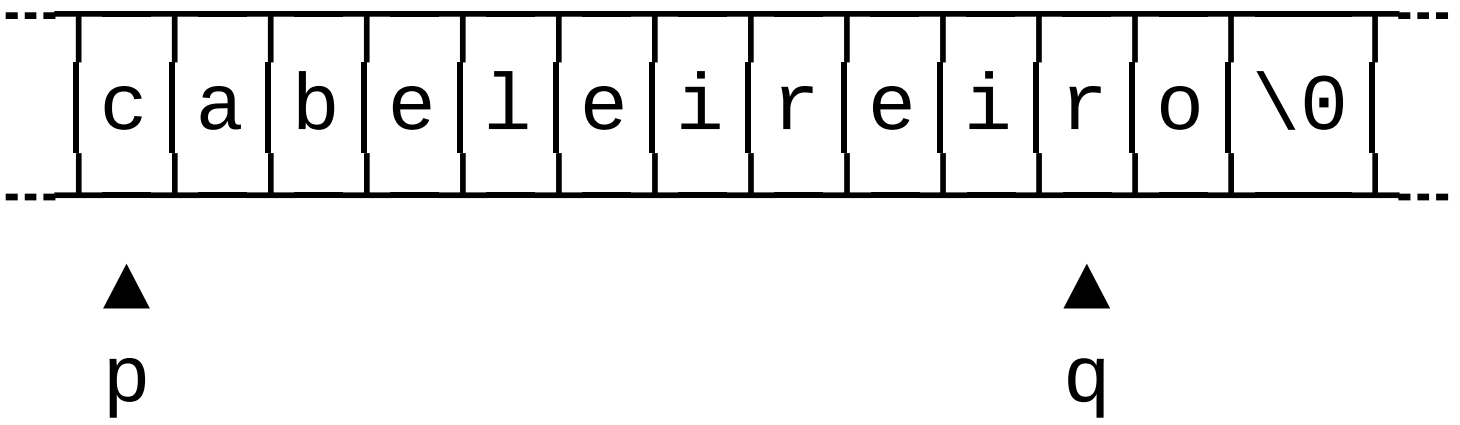
```
mystr_ocorrencias(p, a)
p é "cabeleireiro", a é "ei"
```




```
q = strstr(p, a)
q += strlen(a)
q = strstr(q, a)
q += strlen(a)
```

CONTANDO OCORRENCIAS

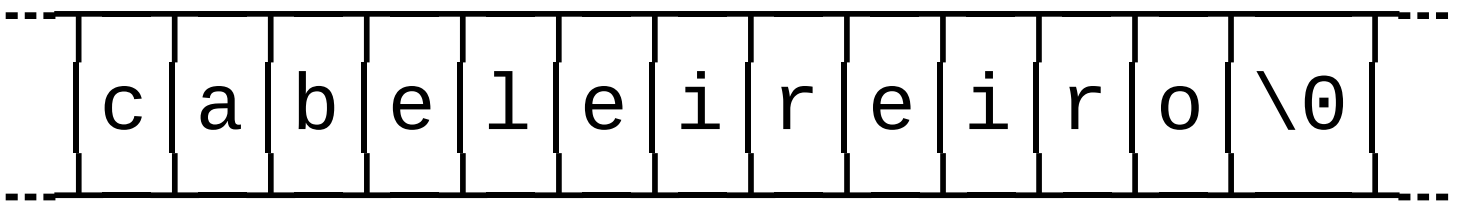
```
mystr_ocorrencias(p, a)
p é "cabeleireiro", a é "ei"
```




```
q = strstr(p, a)
q += strlen(a)
q = strstr(q, a)
q += strlen(a)
q = strstr(p, a)
```

CONTANDO OCORRENCIAS

```
mystr_ocorrencias(p, a)
p é "cabeleireiro", a é "ei"
```



▲

p

q = NULL

```
q = strstr(p, a)
q += strlen(a)
q = strstr(q, a)
q += strlen(a)
q = strstr(p, a)
```

CONTANDO OCORRENCIAS

```
int mystr_ocorrencias(const char *p,
                      const char *a) {
  const char *q = strstr(p, a);
  int ocorrencias = 0;
  while (q != NULL) {
    ++ocorrencias;
    q = strstr(q+strlen(a), a);
  }
  return ocorrencias;
}
```

CONTANDO OCORRENCIAS

```
int mystr_ocorrencias(const char *p,
                      const char *a) {
  p = strstr(p, a);
  int ocorrencias = 0;
  while (p != NULL) {
    ++ocorrencias;
    p = strstr(p+strlen(a), a);
  }
  return ocorrencias;
}
```

USANDO A BIBLIOTECA string.h

- Implemente a função abaixo, que separa uma string em um array, de acordo com o separador fornecido. O array deve ser terminado com NULL.

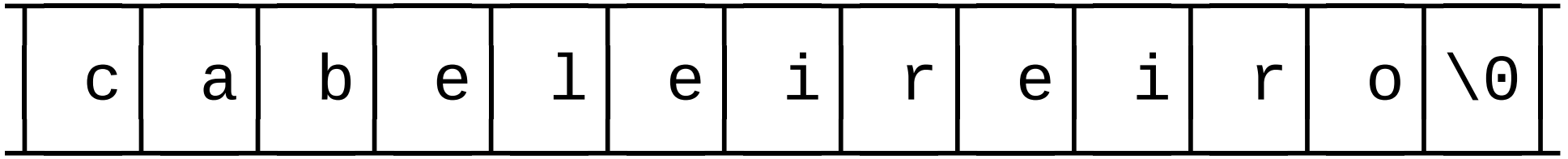
```
char **mystr_split(const char *str,
                   const char *sep)
```

Se o separador for a string vazia "", a função deve funcionar como mystr_explode.

```
mystr_split("cabeleireiro", "ei")
deve retornar o array:
{ "cabel", "r", "ro", NULL }
```

CONTANDO OCORRENCIAS

```
mystr_split("cabeleireiro", "ei")
  char *p = strdup("cabeleireiro");
  char *q = p;
  char **ret = (char **)malloc(...);
```

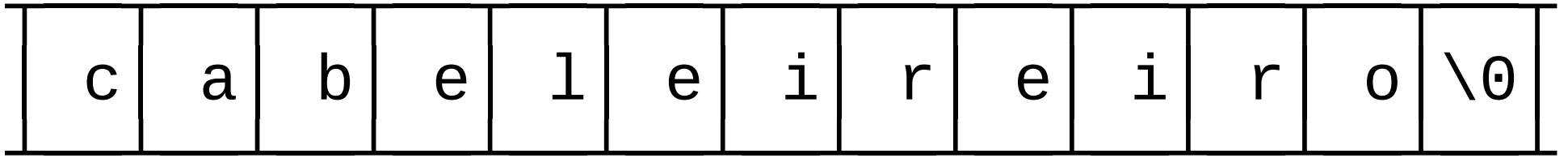


▲

q

CONTANDO OCORRENCIAS

```
mystr_split("cabeleireiro", "ei")
  char *p = strdup("cabeleireiro");
  char *q = p;
  char **ret = (char **)malloc(...);
```



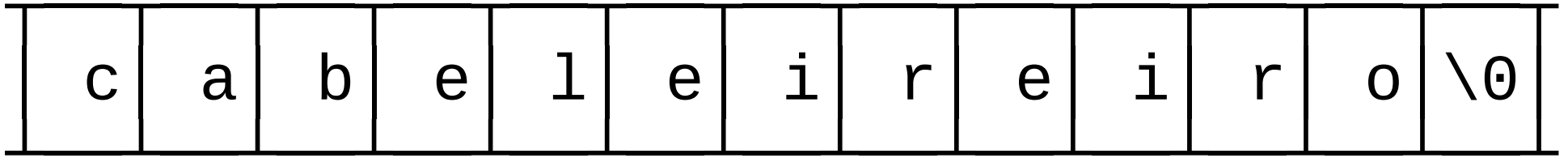
▲

q

```
ret[0] = q;
```

CONTANDO OCORRENCIAS

```
mystr_split("cabeleireiro", "ei")
  char *p = strdup("cabeleireiro");
  char *q = p;
  char **ret = (char **)malloc(...);
```

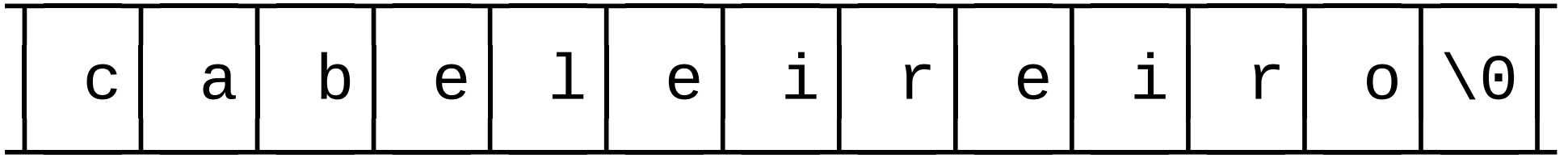


▲

```
q
ret[0] = q;
q = strstr(q, a);
```

CONTANDO OCORRENCIAS

```
mystr_split("cabeleireiro", "ei")
  char *p = strdup("cabeleireiro");
  char *q = p;
  char **ret = (char **)malloc(...);
```



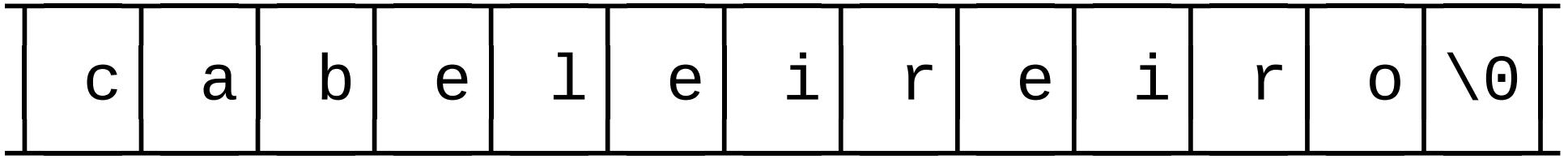
▲

q

```
  ret[0] = q;
  q = strstr(q, a);
```

CONTANDO OCORRENCIAS

```
mystr_split("cabeleireiro", "ei")
  char *p = strdup("cabeleireiro");
  char *q = p;
  char **ret = (char **)malloc(...);
```



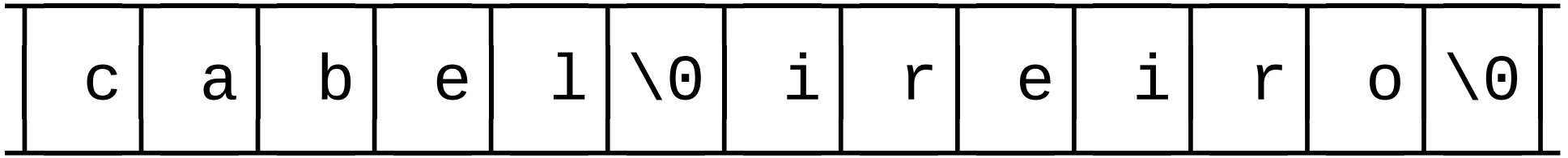
▲

q

```
  ret[0] = q;
  q = strstr(q, a);
  *q = '\0'
```

CONTANDO OCORRENCIAS

```
mystr_split("cabeleireiro", "ei")
  char *p = strdup("cabeleireiro");
  char *q = p;
  char **ret = (char **)malloc(...);
```



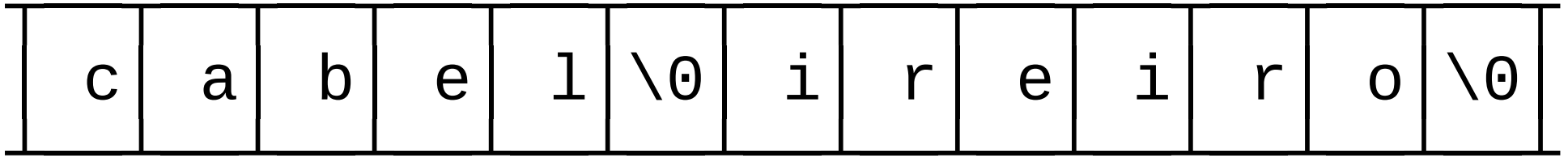
▲

q

```
  ret[0] = q;
  q = strstr(q, a);
  *q = '\0';
```

CONTANDO OCORRENCIAS

```
mystr_split("cabeleireiro", "ei")
  char *p = strdup("cabeleireiro");
  char *q = p;
  char **ret = (char **)malloc(...);
```

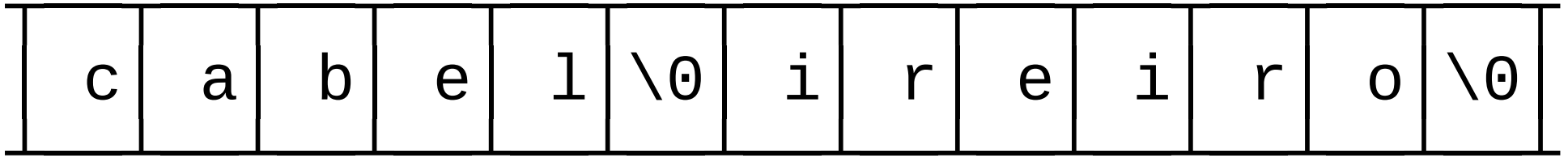


q

```
  ret[0] = q;
  q = strstr(q, a);
  *q = '\0';
  q += strlen(a);
```

CONTANDO OCORRENCIAS

```
mystr_split("cabeleireiro", "ei")
  char *p = strdup("cabeleireiro");
  char *q = p;
  char **ret = (char **)malloc(...);
```

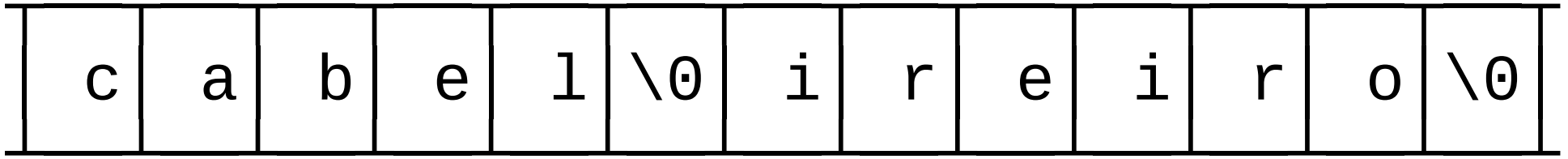


q

```
  ret[0] = q;
  q = strstr(q, a);
  *q = '\0';
  q += strlen(a);
```

CONTANDO OCORRENCIAS

```
mystr_split("cabeleireiro", "ei")
  char *p = strdup("cabeleireiro");
  char *q = p;
  char **ret = (char **)malloc(...);
```

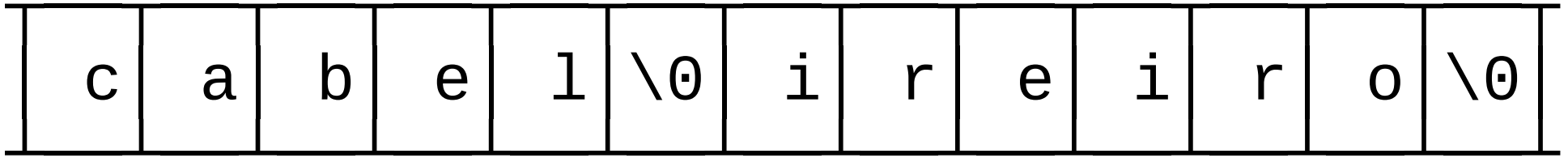


q

```
  ret[0] = q;
  q = strstr(q, a);
  *q = '\0';
  q += strlen(a);
  ret[1] = q;
```

CONTANDO OCORRENCIAS

```
mystr_split("cabeleireiro", "ei")
  char *p = strdup("cabeleireiro");
  char *q = p;
  char **ret = (char **)malloc(...);
```



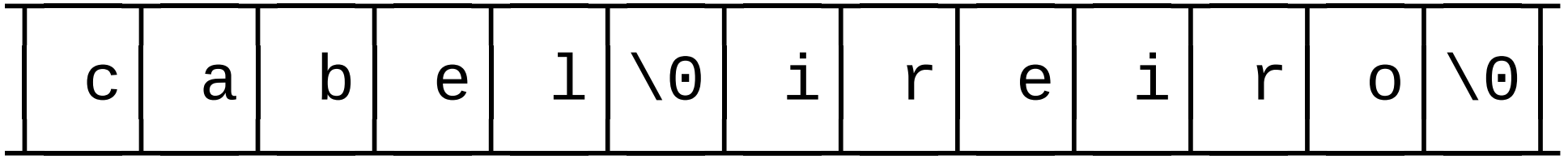
q

```
ret[0] = q;
q = strstr(q, a);
*q = '\0';
q += strlen(a);
ret[1] = q;
```

```
q = strstr(q, a);
```

CONTANDO OCORRENCIAS

```
mystr_split("cabeleireiro", "ei")
  char *p = strdup("cabeleireiro");
  char *q = p;
  char **ret = (char **)malloc(...);
```



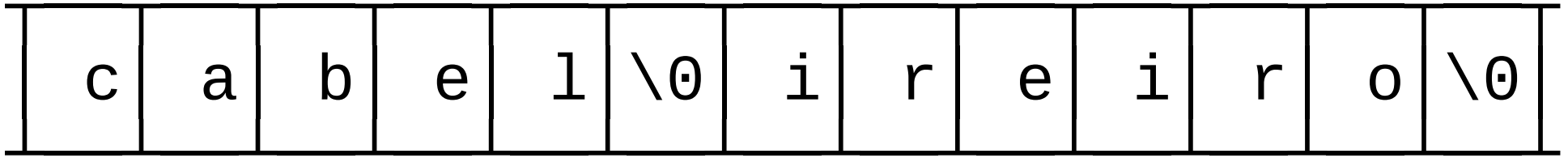
q

```
  ret[0] = q;
  q = strstr(q, a);
  *q = '\0';
  q += strlen(a);
  ret[1] = q;
```

```
q = strstr(q, a);
```

CONTANDO OCORRENCIAS

```
mystr_split("cabeleireiro", "ei")
  char *p = strdup("cabeleireiro");
  char *q = p;
  char **ret = (char **)malloc(...);
```



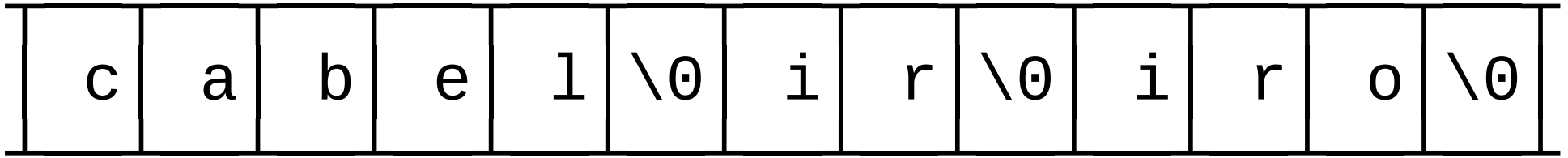
▲
q

```
  ret[0] = q;
  q = strstr(q, a);
  *q = '\0';
  q += strlen(a);
  ret[1] = q;
```

```
q = strstr(q, a);
*q = '\0';
```

CONTANDO OCORRENCIAS

```
mystr_split("cabeleireiro", "ei")
  char *p = strdup("cabeleireiro");
  char *q = p;
  char **ret = (char **)malloc(...);
```



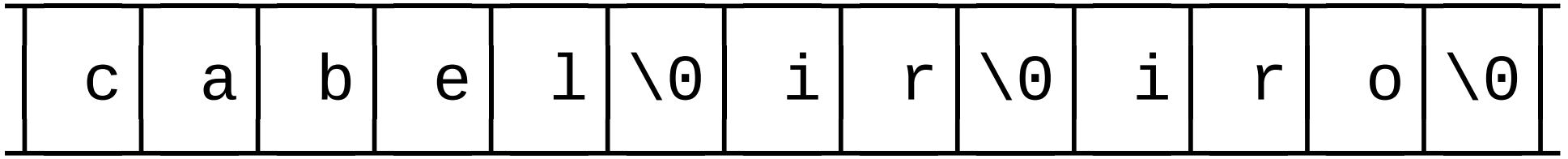
▲
q

```
  ret[0] = q;
  q = strstr(q, a);
  *q = '\0';
  q += strlen(a);
  ret[1] = q;
```

```
q = strstr(q, a);
*q = '\0';
q += strlen(a);
```

CONTANDO OCORRENCIAS

```
mystr_split("cabeleireiro", "ei")
  char *p = strdup("cabeleireiro");
  char *q = p;
  char **ret = (char **)malloc(...);
```



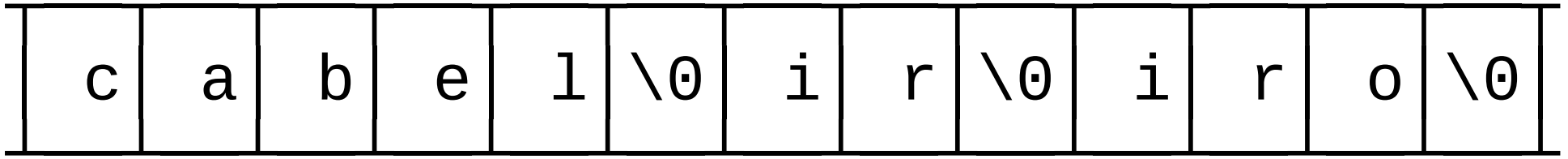
▲
q

```
ret[0] = q;
q = strstr(q, a);
*q = '\0';
q += strlen(a);
ret[1] = q;
```

```
q = strstr(q, a);
*q = '\0';
q += strlen(a);
```

CONTANDO OCORRENCIAS

```
mystr_split("cabeleireiro", "ei")
  char *p = strdup("cabeleireiro");
  char *q = p;
  char **ret = (char **)malloc(...);
```



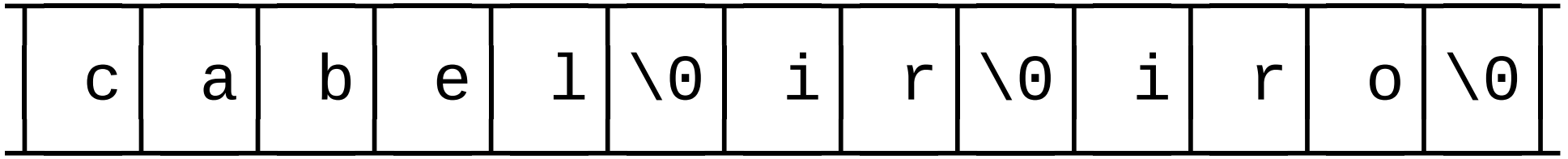
▲
q

```
ret[0] = q;
q = strstr(q, a);
*q = '\0';
q += strlen(a);
ret[1] = q;
```

```
q = strstr(q, a);
*q = '\0';
q += strlen(a);
ret[2] = q;
```

CONTANDO OCORRENCIAS

```
mystr_split("cabeleireiro", "ei")
  char *p = strdup("cabeleireiro");
  char *q = p;
  char **ret = (char **)malloc(...);
```



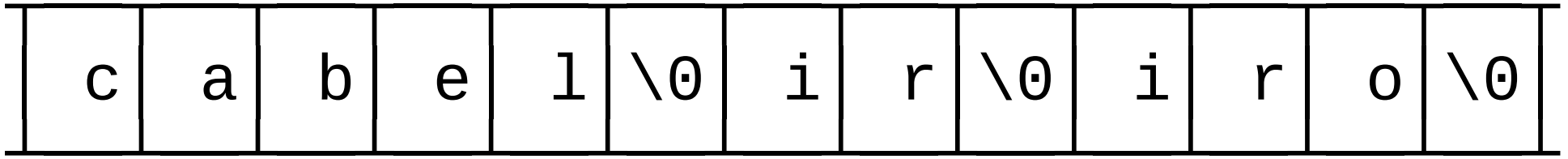
q

```
ret[0] = q;
q = strstr(q, a);
*q = '\0';
q += strlen(a);
ret[1] = q;
```

```
q = strstr(q, a);
*q = '\0';
q += strlen(a);
ret[2] = q;
q = strstr(q, a);
```

CONTANDO OCORRENCIAS

```
mystr_split("cabeleireiro", "ei")
  char *p = strdup("cabeleireiro");
  char *q = p;
  char **ret = (char **)malloc(...);
```

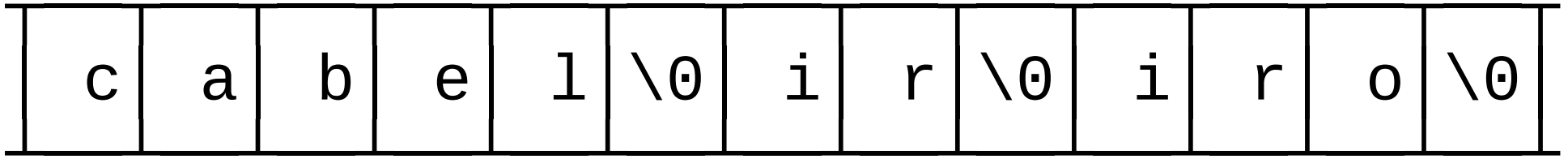



```
  ret[0] = q;
  q = strstr(q, a);
  *q = '\0';
  q += strlen(a);
  ret[1] = q;
```

```
                    q = NULL
  q = strstr(q, a);
  *q = '\0';
  q += strlen(a);
  ret[2] = q;
  q = strstr(q, a);
```

CONTANDO OCORRENCIAS

```
mystr_split("cabeleireiro", "ei")
  char *p = strdup("cabeleireiro");
  char *q = p;
  char **ret = (char **)malloc(...);
```



▲ ret[0]

```
  ret[0] = q;
  q = strstr(q, a);
  *q = '\0';
  q += strlen(a);
  ret[1] = q;
```

▲ ret[1]

▲ ret[2]

```
  q = strstr(q, a);
  *q = '\0';
  q += strlen(a);
  ret[2] = q;
  q = strstr(q, a);
```

```
char **mystr_split(const char *str,
                   const char *sep) {
    if (strlen(sep) == 0)
        return mystr_explode(str);
    char *p = strdup(str);
    int ocorr = mysql_ocorrencias(str, sep);
    char **ret = (char**)malloc((ocorr+2)*
                                sizeof(char*));
    int i = 0;
    ret[i++] = p;
    p = strstr(p, sep);
    while (p != NULL) {
        *p = '\0';
        p += strlen(sep);
        ret[i++] = p;
        p = strstr(p, sep);
    }
    ret[i] = NULL;
    return ret;
}
```

```
char **mystr_split(const char *str,
                   const char *sep) {
    if (strlen(sep) == 0)
        return mystr_explode(str);
    char *p = strdup(str);
    int ocorr = mysql_ocorrencias(str, sep);
    char **ret = (char**)malloc((ocorr+2)*
                                sizeof(char*));
    int i = 0;
    ret[i++] = p;
→   p = strstr(p, sep);
    while (p != NULL) {
        *p = '\0';
        p += strlen(sep);
        ret[i++] = p;
→       p = strstr(p, sep);
    }
    ret[i] = NULL;
    return ret;
}
```

```
char **mystr_split(const char *str,
                   const char *sep) {
    if (strlen(sep) == 0)
        return mystr_explode(str);
    char *p = strdup(str);
    int ocorr = mysql_ocorrencias(str, sep);
    char **ret = (char**)malloc((ocorr+2)*
                                sizeof(char*));
    int i = 0;
    ret[i++] = p;
    while ((p = strstr(p, sep)) != NULL) {
        *p = '\0';
        p += strlen(sep);
        ret[i++] = p;
    }
    ret[i] = NULL;
    return ret;
}
```

```
char **mystr_split(const char *str,
                   const char *sep) {
    if (strlen(sep) == 0)
        return mystr_explode(str);
    char *p = strdup(str);
→   if (p == NULL) return NULL;
    int ocorr = mysql_ocorrencias(str, sep);
    char **ret = (char**)malloc((ocorr+2)*
                                sizeof(char*));
→   if (ret == NULL) {
→       free(p);
→       return NULL;
→   }
    ...
```