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

# Ponteiros e Passagem de Parâmetros

## AULA PASSADA - ponteiros

Uma declaração tal como:

```
tipo *nome;
```

Declara um ponteiro: referência a uma posição de memória onde se encontra uma variável daquele tipo.

Exemplos:

```
int *numeros;

double *notas;

char *nomeDoMes;
```

Obs.: se apenas declararmos o ponteiro, ele indicará uma região indeterminada.

# AULA PASSADA - ponteiros

Podemos declarar um ponteiro, alocar uma região de memória no heap e fazer a referência apontar para essa região alocada:

```
tipo *ptr = (tipo *)malloc(n*sizeof(tipo));
```

O que isto faz:

1) declara ponteiro ptr para **tipo**
2) aloca espaço para n elementos, cada um com tamanho sizeof(**tipo**)
3) (**tipo** *) converte o ponteiro sem tipo retornado por malloc p/ tipo correto
4) faz ptr apontar para o primeiro elemento da região

# PONTEIROS – exemplo exptr.c

```c
#include <stdlib.h> // para usar malloc
#include <stdio.h> // para usar puts

int main(void) {
  char *p = (char*)malloc(3*sizeof(char));
  if (p == NULL) return 1;
  p[0] = 'o';
  p[1] = 'i';
  p[2] = '\0';
  puts(p);
  p[0] = 'h';
  puts(p);
  free(p);
}
```

# PONTEIROS – exemplo exptr.c

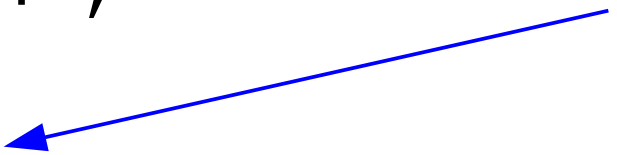
```
#include <stdlib.h> // para usar malloc
#include <stdio.h> // para usar puts

int main(void) {
  char *p = (char*)malloc(3*sizeof(char));
  if (p == NULL) return 1;
  p[0] = 'o';
  p[1] = 'i';
  p[2] = '\0';
  puts(p);
  p[0] = 'h';
  puts(p);
  free(p);
}
```

*Sempre verifique se o ponteiro retornado por* `malloc` *é válido!*

# PONTEIROS – exemplo exptr.c

```
#include <stdlib.h> // para usar malloc
#include <stdio.h> // para usar puts

int main(void) {
  char *p = (char*)malloc(3*sizeof(char));
  if (p == NULL) return 1;
  p[0] = 'o';
  p[1] = 'i';
  p[2] = '\0';
  puts(p);
  p[0] = 'h';
  puts(p);
  free(p);
}
```

*Após o último uso, sempre libere a área alocada.*



# PONTEIROS – exemplo exptr.c

```
#include <stdlib.h> // para usar malloc
#include <stdio.h> // para usar puts

int main(void) {
  char *p = (char*)malloc(3*sizeof(char));
  if (p == NULL) return 1;
  p[0] = 'o';
  p[1] = 'i';
  p[2] = '\0';
  puts(p);
  p[0] = 'h';
  puts(p);
  free(p)
}
```

*Saída:*

```
oi
hi
```

## PONTEIROS – exemplo exptr**2**.c

Edite e compile o programa exptr**2**.c:

```
#include <stdio.h> // para usar puts

int main(void) {
  char str[] = "oi";
  char *p = str;
  puts(str);
  puts(p);
  p[0] = 'h';
  puts(str);
  puts(p);
}
```

Explique o comportamento do programa.

# PONTEIROS – exemplo exptr**2**.c

Edite e compile o programa exptr**2**.c:

```
#include <stdio.h> // para usar puts

int main(void) {
  char str[] = "oi";
  char *p = str;
  puts(str);
  puts(p);
  p[0] = 'h';
  puts(str);
  puts(p);
}
```

*Saída:*

```
oi
oi
hi
hi
```

Explique o comportamento do programa.

# PONTEIROS – exemplo exptr**2**.c

Edite e compile o programa exptr**2**.c:

```
#include <stdio.h> // para usar puts

int main(void) {
   char str[] = "oi";
   char *p = str;
   puts(str);
   puts(p);
   p[0] = 'h';
   puts(str);
   puts(p);
}
```

*Saída:*

```
oi
oi
hi
hi
```

p e str apontam para a mesma região de mem.

## PONTEIROS – exemplo exptr**3**.c

Edite e compile o programa exptr**3**.c:

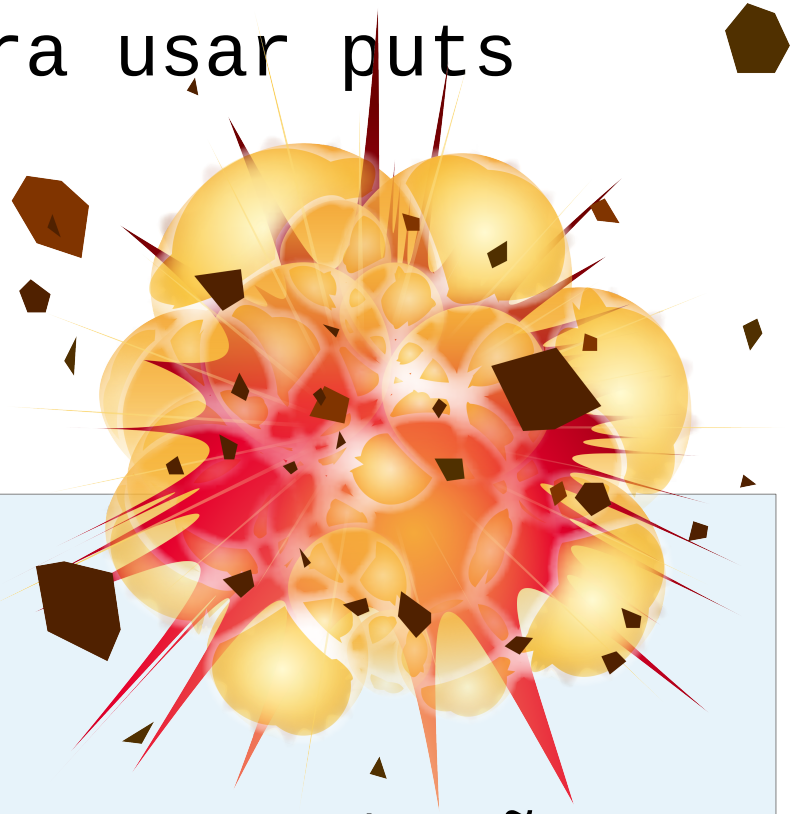
```
#include <stdio.h> // para usar puts

int main(void) {
  char *str = "oi";
  char *p = str;
  puts(str);
  puts(p);
  p[0] = 'h';
  puts(str);
  puts(p);
}
```

O que houve?

# PONTEIROS - exemplo exptr**3**.c

Edite e compile o programa exptr**3**.c:

```
#include <stdio.h> // para usar puts

int main(void) {
   char *str = "oi";
   char *p = str;
   puts(str);
   puts(p);
   p[0] = 'h';
   puts(str);
   puts(p);
}
```



*Saída:*

```
oi
oi
Falha de segmentação
```

O que houve?

## PONTEIROS - exemplo exptr**3**.c

```
int main(void) {
   char *str = "oi";
   char *p = str;
   puts(str);
   puts(p);
   p[0] = 'h';
   puts(str);
   puts(p);
}
```

*faz* `str` *apontar para a* ***constante*** *"oi"*

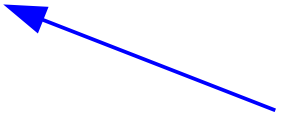
## PONTEIROS - exemplo exptr**3**.c

```
int main(void) {
   char *str = "oi";
   char *p = str;
   puts(str);
   puts(p);
   p[0] = 'h';
   puts(str);
   puts(p);
}
```

*faz* p *apontar para a mesma região de memória que* `str` *(lembre: a região de memória é* ***imutável*** *pois guarda uma* ***constante****)*

## PONTEIROS - exemplo exptr**3**.c

```
int main(void) {
   char *str = "oi";
   char *p = str;
   puts(str);
   puts(p);
   p[0] = 'h';
   puts(str);
   puts(p);
}
```



*imprime a string apontada por* `str`

## PONTEIROS - exemplo exptr**3**.c

```
int main(void) {
   char *str = "oi";
   char *p = str;
   puts(str);
   puts(p);
   p[0] = 'h';
   puts(str);
   puts(p);
}
```

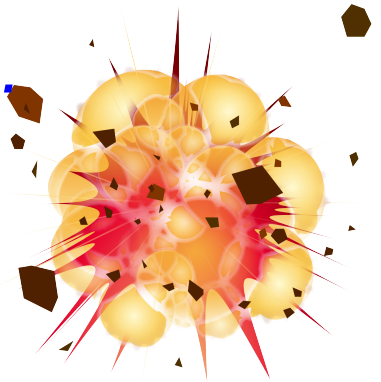
*imprime a string apontada por* p

# PONTEIROS - exemplo exptr**3**.c

```
int main(void) {
   char *str = "oi";
   char *p = str;
   puts(str);
   puts(p);
   p[0] = 'h';
   puts(str);
   puts(p);
}
```

*tenta alterar o primeiro caractere da região apontada por* p

*como a região apontada por* p *faz parte de área de memória* ***imutável****...*



## Moral dos exemplos exptr**1**.c a exptr**3**.c

- Um ponteiro pode indicar uma área de memória no heap, na pilha **ou em outras regiões de memória** (se houver), **que podem ser imutáveis**
- Declara ponteiro e aloca espaço no **heap** (deve ser desalocado c/ free):
  `char *p = (char *)malloc(tamanho);`
- Declara ponteiro e aloca espaço na **pilha** (desalocação automática):
  `char p[] = "`*inicialização*`";`
- Declara ponteiro e aponta p/ uma constante (região imutável):
  `char *p = "`*inicialização*`";`

## PONTEIROS - exemplo exptr**4**.c

```c
#include <stdio.h>

int main(void) {
   int i = 42;
   int *p = &i;

   printf("i é %d\n", i);
   p[0]++;
   printf("i é %d\n", i);
}
```

# PONTEIROS - exemplo exptr4.c

```c
#include <stdio.h>

int main(void) {
   int i = 42;
   int *p = &i;

   printf("i é %d\n", i);
   p[0]++;
   printf("i é %d\n", i);
}
```

***Operador &***

*retorna referência (endereço) da posição de memória onde está armazenada a variável*

# PONTEIROS - exemplo exptr**4**.c

```c
#include <stdio.h>

int main(void) {
   int i = 42;
   int *p = &i;

   printf("i é %d\n", i);
   p[0]++;
   printf("i é %d\n", i);
}
```

*Saída:*

```
i é 42
i é 43
```

Por que o valor de i mudou? Por que não fez “cabum”?

## PONTEIROS - exemplo exptr**5**.c

```c
#include <stdio.h>

int main(void) {
   int i[] = { 42, 99 };
   int *p = i;

   printf("i[0] é %d\n", i[0]);
   p[0]++;
   printf("i[0] é %d\n", i[0]);
}
```

Funciona?

## PONTEIROS - exemplo exptr**6**.c

```c
#include <stdio.h>

int main(void) {
  static const int i[] = { 42, 99 };
  int *p = i;

  printf("i[0] é %d\n", i[0]);
  p[0]++;
  printf("i[0] é %d\n", i[0]);
}
```

Funciona?

## OPERAÇÕES com PONTEIROS

**Operador de referência &**

Colocado na frente de uma variável, nos dá o endereço onde esta variável está armazenada.

```
int i = 42;
int *p = &i;

p[0]++; // incrementa o primeiro int
        // na região de memória apontada
        // por p
```

# OPERAÇÕES com PONTEIROS

**Operador de referência &**

Colocado na frente de uma variável, nos dá o endereço onde esta variável está armazenada.

Por que não se usa o operador & abaixo?

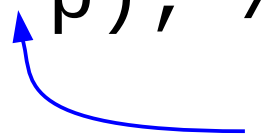
```
int i[] = { 42, 99 };
int *p = i;

p[0]++;
```

## OPERAÇÕES com PONTEIROS

### Operador de dereferência *

Colocado na frente de uma **variável ou expressão**, permite acessar o **conteúdo** da posição de memória apontada por essa variável ou expressão.

```
int i = 42;
int *p = &i;

++(*p); // equivale a ++p[0]
```



***Atenção!*** *Neste exmplo, apenas este uso do caractere * indica o operador de dereferência!*

## OPERAÇÕES com PONTEIROS

**Operador de dereferência ***

Infelizmente, a notação é confusa. O caractere * indica dereferência **se não vier imediatamente precedido por um tipo.**

```
int *p; // declara variável do tipo
        // ponteiro para int

*p = 42; // coloca na posição de memória
         // apontada por p o número 42
```

## Experimento no **cling**

Inicie o cling (se já tiver iniciado, saia com .q e reinicie-o)

```
[cling]$ #include <stdio.h>
[cling]$ int a[] = { 18, 33, 99 };
[cling]$ int b[] = { 42, 21 };
```

## Experimento no **cling**

Inicie o cling (se já tiver iniciado, saia com .q e reinicie-o)

```
[cling]$ #include <stdio.h>
[cling]$ int a[] = { 18, 33, 99 };
[cling]$ int b[] = { 42, 21 };
[cling]$ int *p = a;
[cling]$ int *q = b;
```

## Experimento no **cling**

Inicie o cling (se já tiver iniciado, saia com .q e reinicie-o)

```
[cling]$ #include <stdio.h>
[cling]$ int a[] = { 18, 33, 99 };
[cling]$ int b[] = { 42, 21 };
[cling]$ int *p = a;
[cling]$ int *q = b;
[cling]$ printf("%d\n", p);
```

## Experimento no **cling**

Inicie o cling (se já tiver iniciado, saia com .q e reinicie-o)

```
[cling]$ #include <stdio.h>
[cling]$ int a[] = { 18, 33, 99 };
[cling]$ int b[] = { 42, 21 };
[cling]$ int *p = a;
[cling]$ int *q = b;
[cling]$ printf("%d\n", p);
```

*Warning: … (tentando imprimir um ponteiro com “%d”)*

## Experimento no **cling**

Inicie o cling (se já tiver iniciado, saia com .q e reinicie-o)

```
[cling]$ #include <stdio.h>
[cling]$ int a[] = { 18, 33, 99 };
[cling]$ int b[] = { 42, 21 };
[cling]$ int *p = a;
[cling]$ int *q = b;
[cling]$ printf("%d\n", p);
```

*Warning: … (tentando imprimir um ponteiro com “%d”)*

```
[cling]$ printf("%d\n", *p);
```

## Experimento no **cling**

Inicie o cling (se já tiver iniciado, saia com .q e reinicie-o)

```
[cling]$ #include <stdio.h>
[cling]$ int a[] = { 18, 33, 99 };
[cling]$ int b[] = { 42, 21 };
[cling]$ int *p = a;
[cling]$ int *q = b;
[cling]$ printf("%d\n", p);
```

*Warning: … (tentando imprimir um ponteiro com “%d”)*

```
[cling]$ printf("%d\n", *p);
18
```

# Experimento no **cling**

Inicie o cling (se já tiver iniciado, saia com .q e reinicie-o)

```
[cling]$ #include <stdio.h>
[cling]$ int a[] = { 18, 33, 99 };
[cling]$ int b[] = { 42, 21 };
[cling]$ int *p = a;
[cling]$ int *q = b;
[cling]$ printf("%d\n", p);
```

*Warning: … (tentando imprimir um ponteiro com “%d”)*

```
[cling]$ printf("%d\n", *p);
18
[cling]$ printf("%d\n", *q);
```

## Experimento no **cling**

Inicie o cling (se já tiver iniciado, saia com .q e reinicie-o)

```
[cling]$ #include <stdio.h>
[cling]$ int a[] = { 18, 33, 99 };
[cling]$ int b[] = { 42, 21 };
[cling]$ int *p = a;
[cling]$ int *q = b;
[cling]$ printf("%d\n", p);
```

*Warning: … (tentando imprimir um ponteiro com “%d”)*

```
[cling]$ printf("%d\n", *p);
18
[cling]$ printf("%d\n", *q);
42
```

# Experimento no **cling**

Inicie o cling (se já tiver iniciado, saia com .q e reinicie-o)

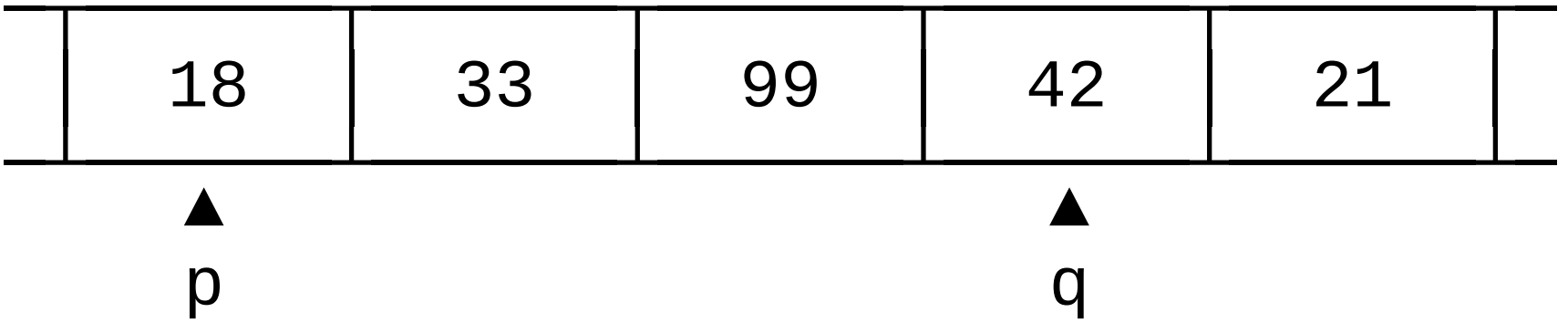
```
[cling]$ #include <stdio.h>
[cling]$ int a[] = { 18, 33, 99 };
[cling]$ int b[] = { 42, 21 };
[cling]$ int *p = a;
[cling]$ int *q = b;
[cling]$ printf("%d\n", p);
Warning: ... (tentando imprimir um ponteiro com "%d")
[cling]$ printf("%d\n", *p);
18
[cling]$ printf("%d\n", *q);
42
```

Permaneça no cling!
Já voltaremos a ele...

# ARITMÉTICA DE PONTEIROS

Podemos fazer contas aritméticas e apontar para outras regiões de memória.

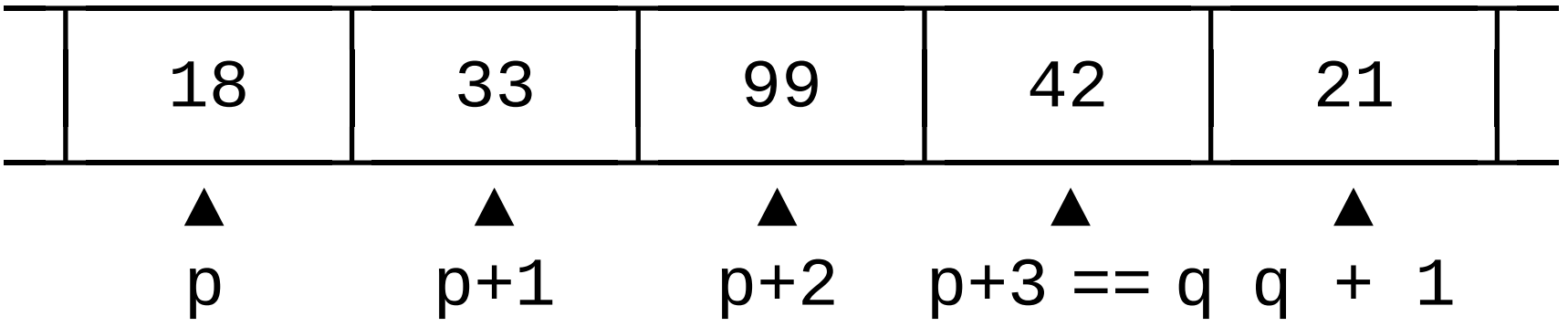
```
[cling]$ int a[]={ 18,33,99 }; int*p=a;
[cling]$ int b[] = { 42, 21 }; int*q=b;
[cling]$ q-p
3
[cling]$ printf("end. de p: %p\n", p);
[cling]$ printf("end. de q: %p\n", q);
```



# ARITMÉTICA DE PONTEIROS

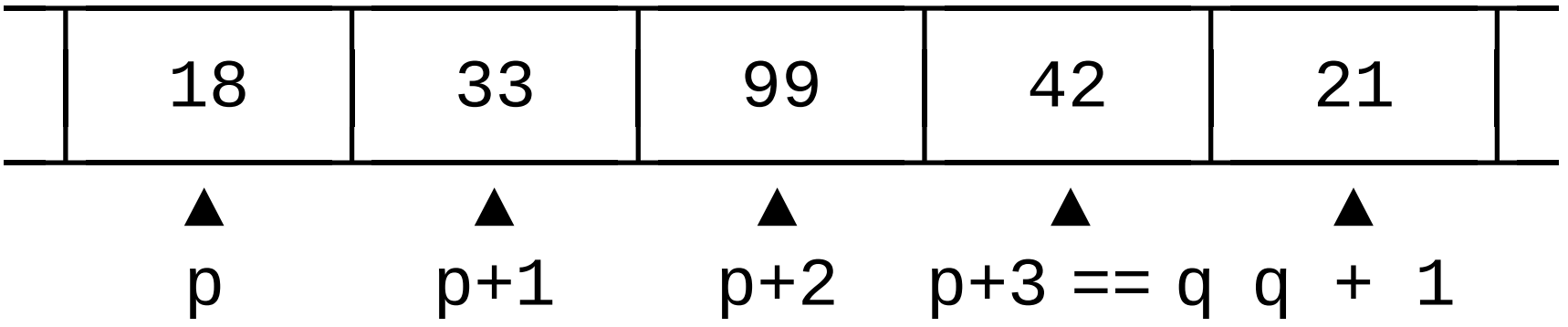
Podemos fazer contas aritméticas e apontar para outras regiões de memória.

```
[cling]$ int a[]={ 18,33,99 }; int*p=a;
[cling]$ int b[] = { 42, 21 }; int*q=b;
[cling]$ q-p
3
[cling]$ printf("end. de p: %p\n", p);
[cling]$ printf("end. de q: %p\n", q);
```



# ARITMÉTICA DE PONTEIROS

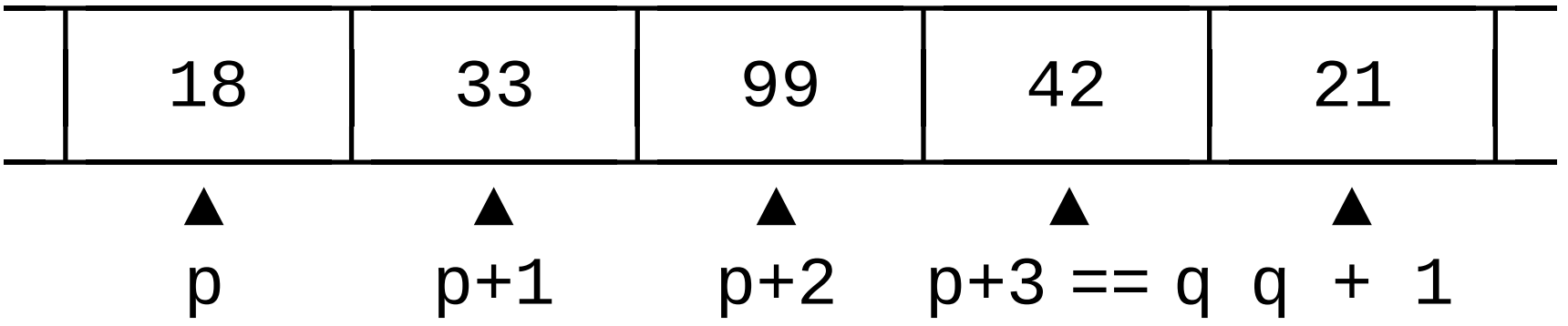
```
[cling]$ *p
(int) 18
[cling]$ *q
(int) 42
[cling]$ *(p+2)
(int) 99
[cling]$ *(p+3)
(int) 42
```



# ARITMÉTICA DE PONTEIROS

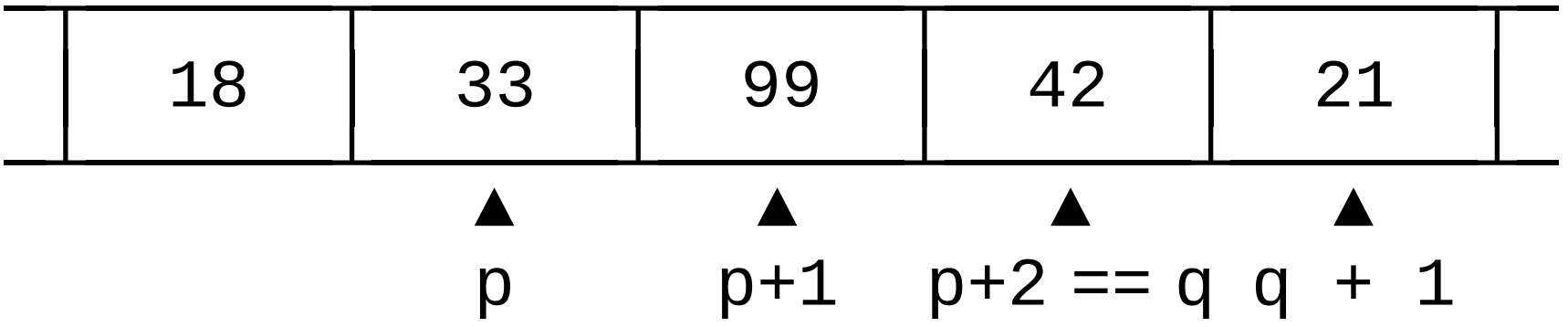
```
[cling]$ *(++p)
```

*Não pressione enter ainda!*
*O que isto faz?*
*Qual será o resultado?*



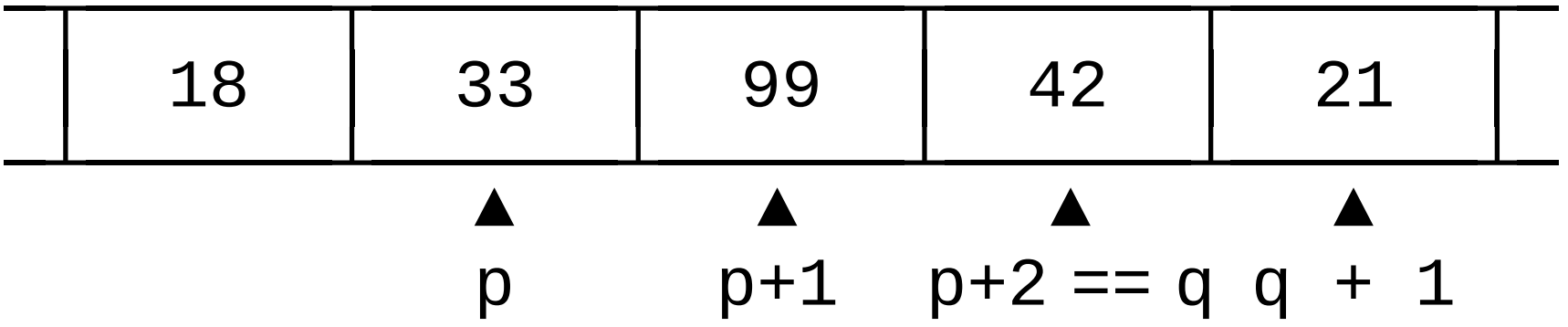
# ARITMÉTICA DE PONTEIROS

```
[cling]$ *(++p)
(int) 33
[cling]$
```



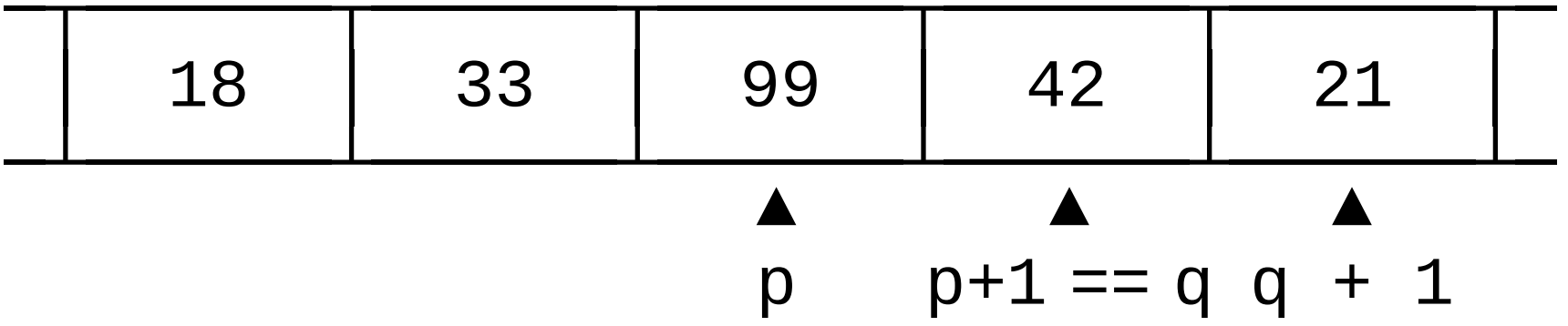
# ARITMÉTICA DE PONTEIROS

```
[cling]$ *(++p)
(int) 33
[cling]$ *(p++)
```



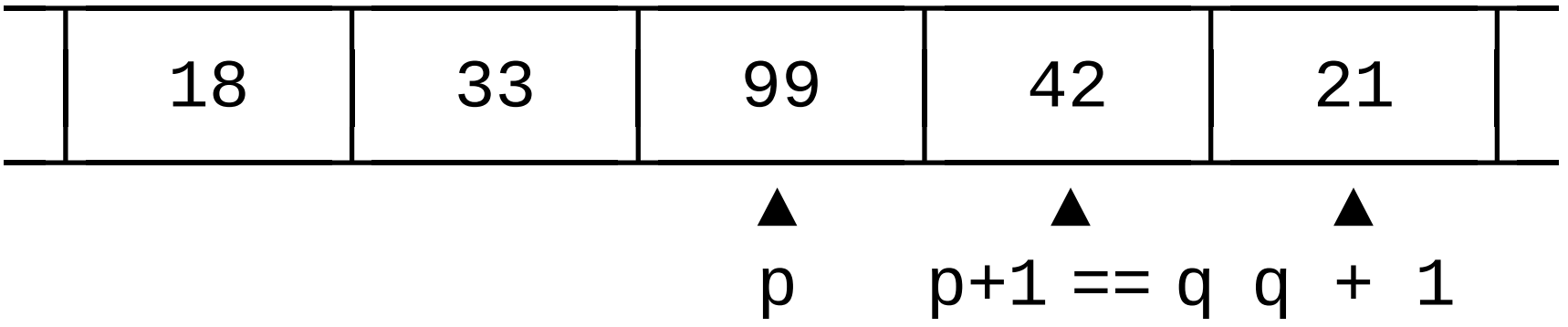
# ARITMÉTICA DE PONTEIROS

```
[cling]$ *(++p)
(int) 33
[cling]$ *(p++)
(int) 33
[cling]$
```



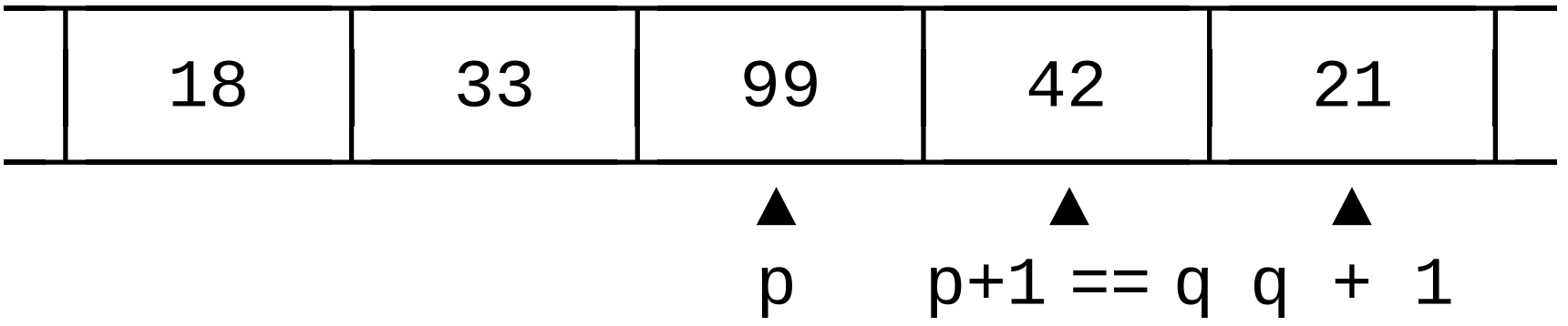
# ARITMÉTICA DE PONTEIROS

```
[cling]$ *(++p)
(int) 33
[cling]$ *(p++)
(int) 33
[cling]$ *p
```



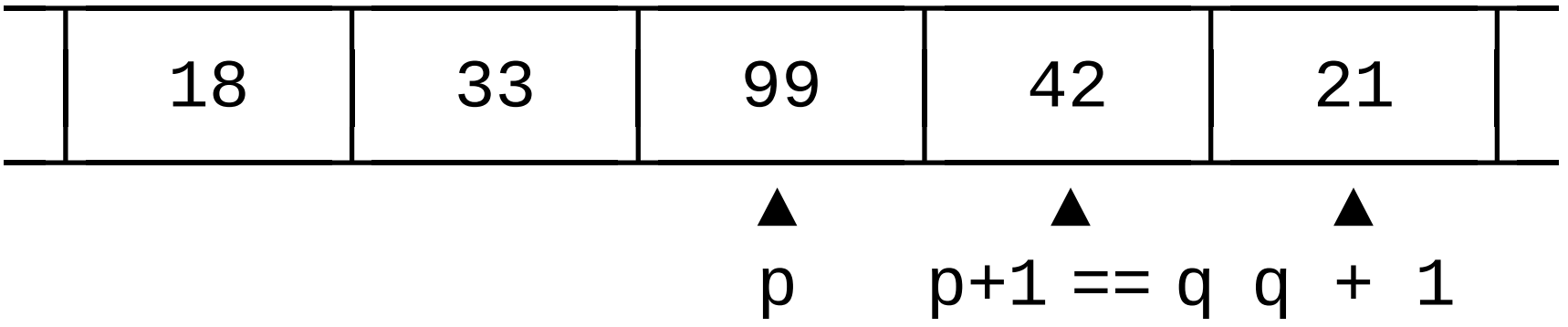
# ARITMÉTICA DE PONTEIROS

```
[cling]$ *(++p)
(int) 33
[cling]$ *(p++)
(int) 33
[cling]$ *p
(int) 99
[cling]$
```



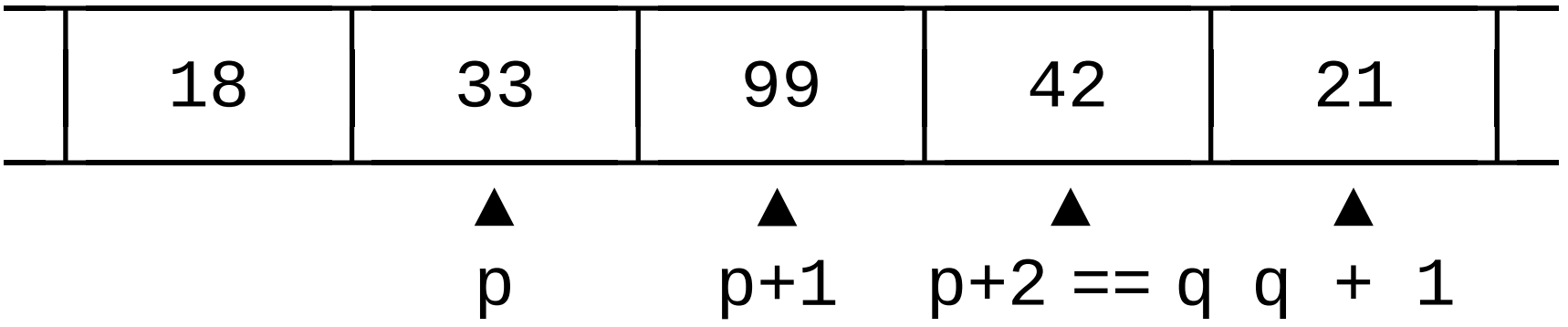
# ARITMÉTICA DE PONTEIROS

```
[cling]$ *(++p)
(int) 33
[cling]$ *(p++)
(int) 33
[cling]$ *p
(int) 99
[cling]$ *(--p)
```



# ARITMÉTICA DE PONTEIROS

```
[cling]$ *(++p)
(int) 33
[cling]$ *(p++)
(int) 33
[cling]$ *p
(int) 99
[cling]$ *(--p)
(int) 33
[cling]$
```



## ARITMÉTICA DE PONTEIROS – palindromo.c

```
int palindromo(const char *p) {




}
```

## ARITMÉTICA DE PONTEIROS – palindromo.c

```
int palindromo(const char *p) {
  const char *q = p;




}
```

# ARITMÉTICA DE PONTEIROS – palindromo.c

```
int palindromo(const char *p) {
  const char *q = p;
  // localiza final da string
  while (*q != '\0') ++q;




}
```

## ARITMÉTICA DE PONTEIROS – palindromo.c

```
int palindromo(const char *p) {
  const char *q = p;
  // localiza final da string
  while (*q != '\0') ++q;
  --q; // q aponta para último char
       // diferente de '\0'




}
```

## ARITMÉTICA DE PONTEIROS – palindromo.c

```
int palindromo(const char *p) {
  const char *q = p;
  // localiza final da string
  while (*q != '\0') ++q;
  --q; // q aponta para último char
       // diferente de '\0'
  // compara
  while (p < q) {


  }

}
```

## ARITMÉTICA DE PONTEIROS – palindromo.c

```
int palindromo(const char *p) {
  const char *q = p;
  // localiza final da string
  while (*q != '\0') ++q;
  --q; // q aponta para último char
       // diferente de '\0'
  // compara
  while (p < q) {
    if (*p != *q) return 0; // falso

  }

}
```

## ARITMÉTICA DE PONTEIROS – palindromo.c

```
int palindromo(const char *p) {
  const char *q = p;
  // localiza final da string
  while (*q != '\0') ++q;
  --q; // q aponta para último char
       // diferente de '\0'
  // compara
  while (p < q) {
    if (*p != *q) return 0; // falso
    ++p; --q;
  }

}
```

## ARITMÉTICA DE PONTEIROS – palindromo.c

```
int palindromo(const char *p) {
  const char *q = p;
  // localiza final da string
  while (*q != '\0') ++q;
  --q; // q aponta para último char
       // diferente de '\0'
  // compara
  while (p < q) {
    if (*p != *q) return 0; // falso
    ++p; --q;
  }
  return 1; // verdadeiro
}
```

# ARITMÉTICA DE PONTEIROS – palindromo.c

Reinicie o cling e teste:

```
[cling]$ .L palindromo.c
[cling]$ palindromo("roma amor")
```

# ARITMÉTICA DE PONTEIROS – palindromo.c

Reinicie o cling e teste:

```
[cling]$ .L palindromo.c
[cling]$ palindromo("roma amor")
(int) 1    // verdadeiro
[cling]$
```

# ARITMÉTICA DE PONTEIROS – palindromo.c

Reinicie o cling e teste:

```
[cling]$ .L palindromo.c
[cling]$ palindromo("roma amor")
(int) 1    // verdadeiro
[cling]$ palindromo("anemona")
```

# ARITMÉTICA DE PONTEIROS – palindromo.c

Reinicie o cling e teste:

```
[cling]$ .L palindromo.c
[cling]$ palindromo("roma amor")
(int) 1    // verdadeiro
[cling]$ palindromo("anemona")
(int) 0    // falso
[cling]$
```

# ARITMÉTICA DE PONTEIROS – palindromo.c

Reinicie o cling e teste:

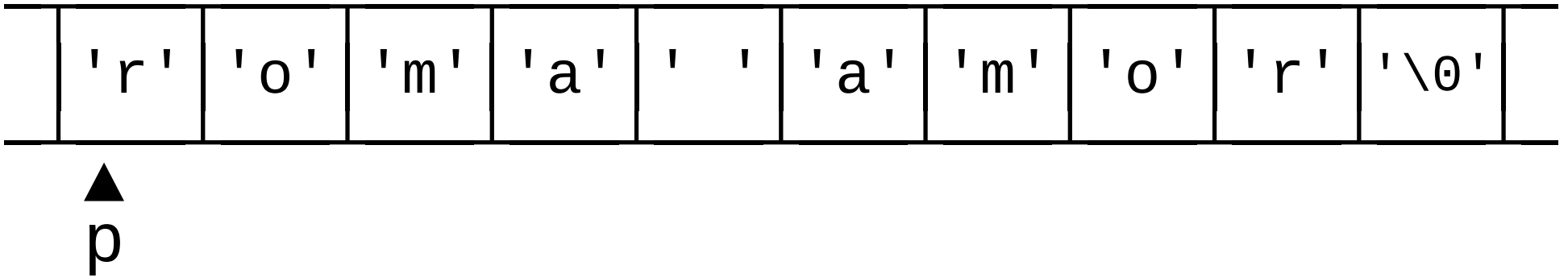
```
[cling]$ .L palindromo.c
[cling]$ palindromo("roma amor")
(int) 1    // verdadeiro
[cling]$ palindromo("anemona")
(int) 0    // falso
[cling]$ palindromo("")
```

# ARITMÉTICA DE PONTEIROS – palindromo.c

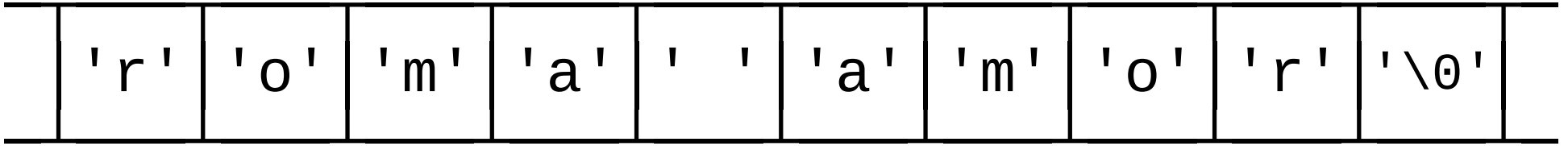
Reinicie o cling e teste:

```
[cling]$ .L palindromo.c
[cling]$ palindromo("roma amor")
(int) 1    // verdadeiro
[cling]$ palindromo("anemona")
(int) 0    // falso
[cling]$ palindromo("")
(int) 1    // verdadeiro
[cling]$
```

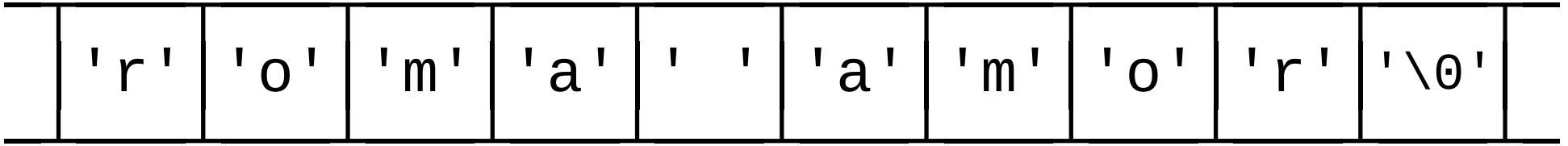
A string vazia é palíndroma, pois equivale a ela mesma lida de trás para frente.



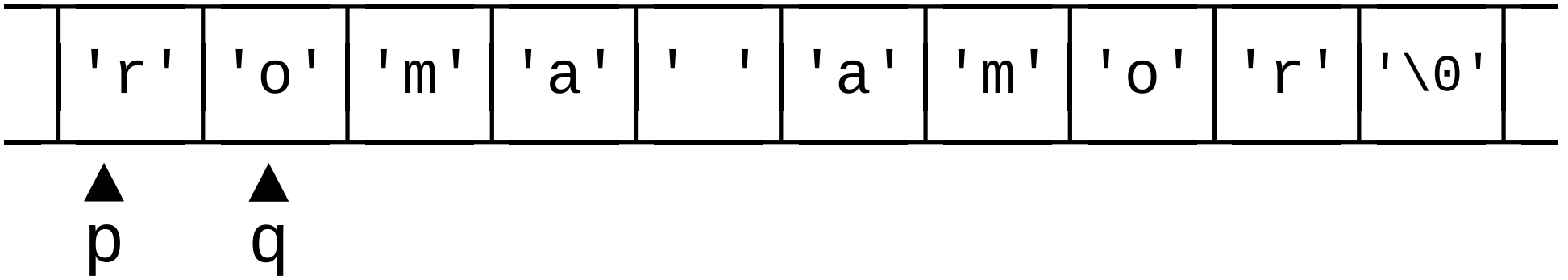

```
int palindromo(const char *p) {
  const char *q = p;

  while (*q != '\0') ++q;
  --q;

  while (p < q) {
    if (*p != *q) return 0;
    ++p; --q;
  }
  return 1;
}
```



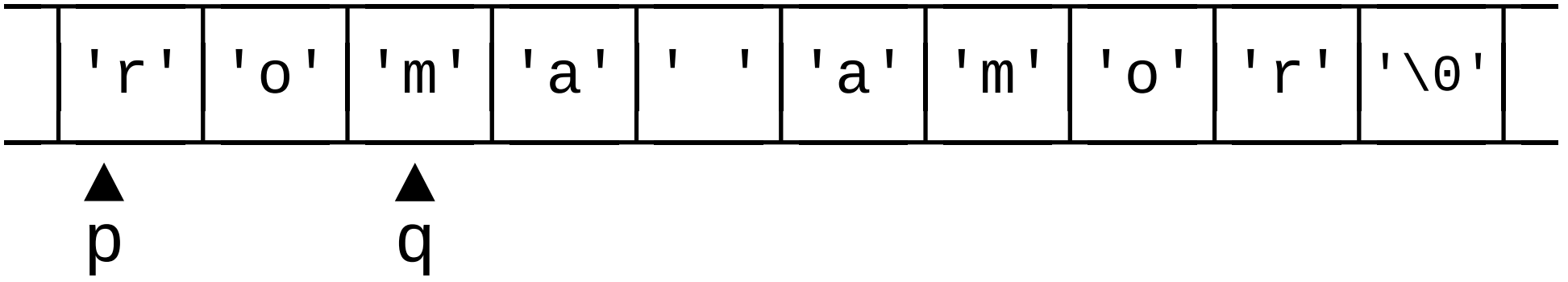

```
int palindromo(const char *p) {
  const char *q = p;

  while (*q != '\0') ++q;
  --q;

  while (p < q) {
    if (*p != *q) return 0;
    ++p; --q;
  }
  return 1;
}
```



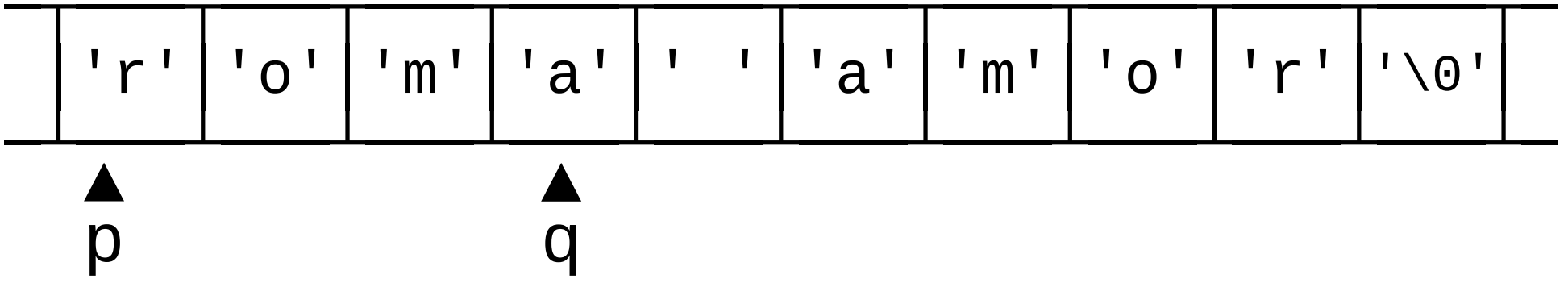

```
int palindromo(const char *p) {
  const char *q = p;

  while (*q != '\0') ++q;
  --q;

  while (p < q) {
    if (*p != *q) return 0;
    ++p; --q;
  }
  return 1;
}
```



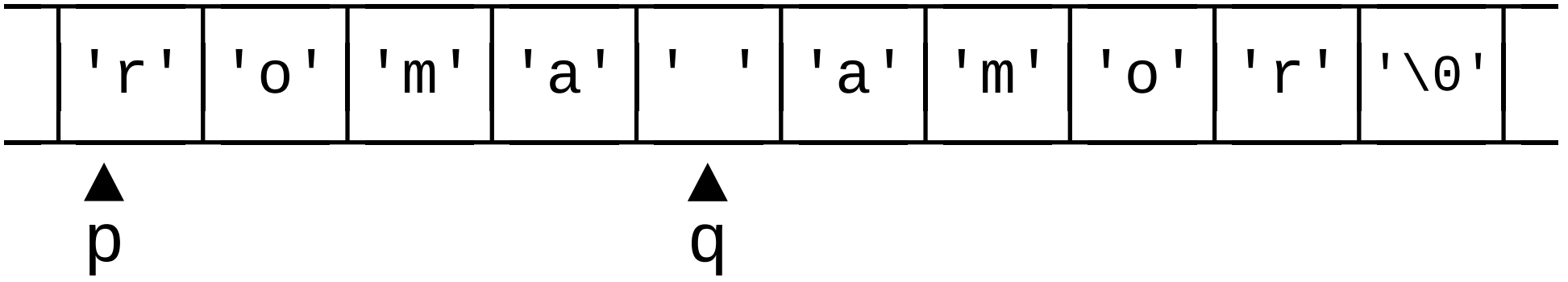

```
int palindromo(const char *p) {
  const char *q = p;

  while (*q != '\0') ++q;
  --q;

  while (p < q) {
    if (*p != *q) return 0;
    ++p; --q;
  }
  return 1;
}
```



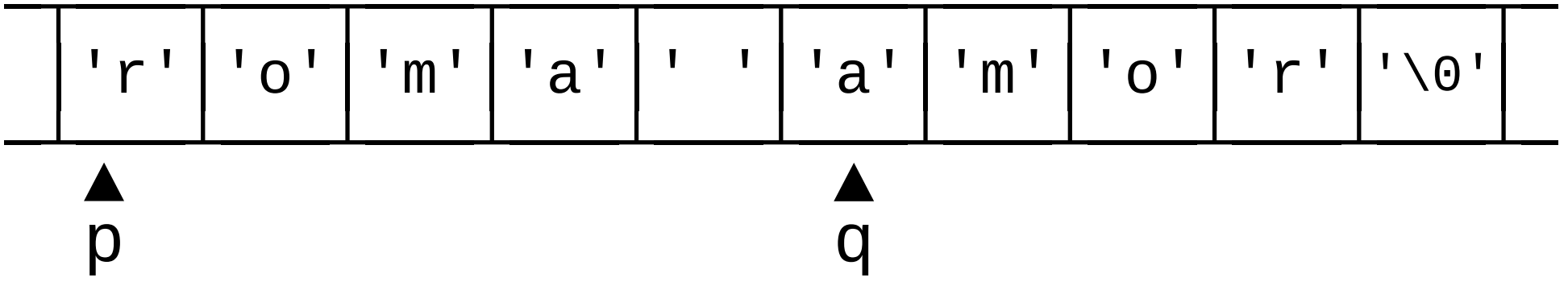

```
int palindromo(const char *p) {
  const char *q = p;

  while (*q != '\0') ++q;
  --q;

  while (p < q) {
    if (*p != *q) return 0;
    ++p; --q;
  }
  return 1;
}
```



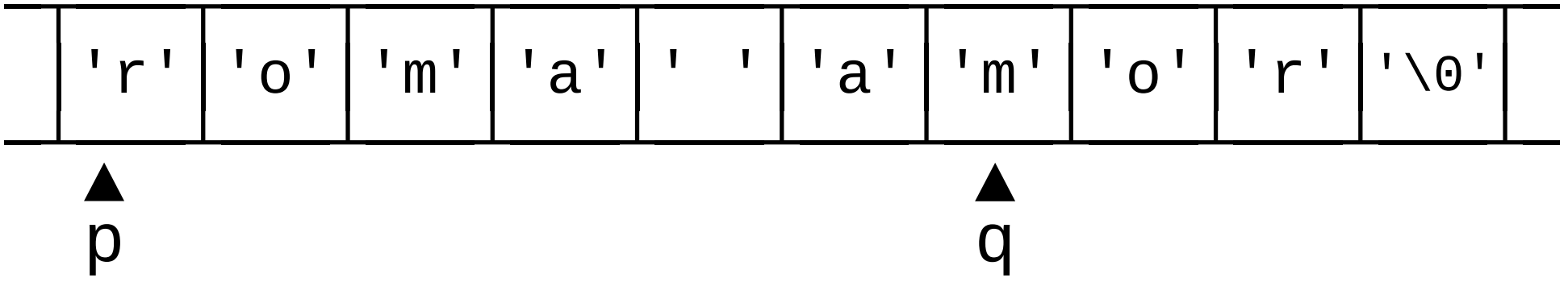

```
int palindromo(const char *p) {
  const char *q = p;

  while (*q != '\0') ++q;
  --q;

  while (p < q) {
    if (*p != *q) return 0;
    ++p; --q;
  }
  return 1;
}
```



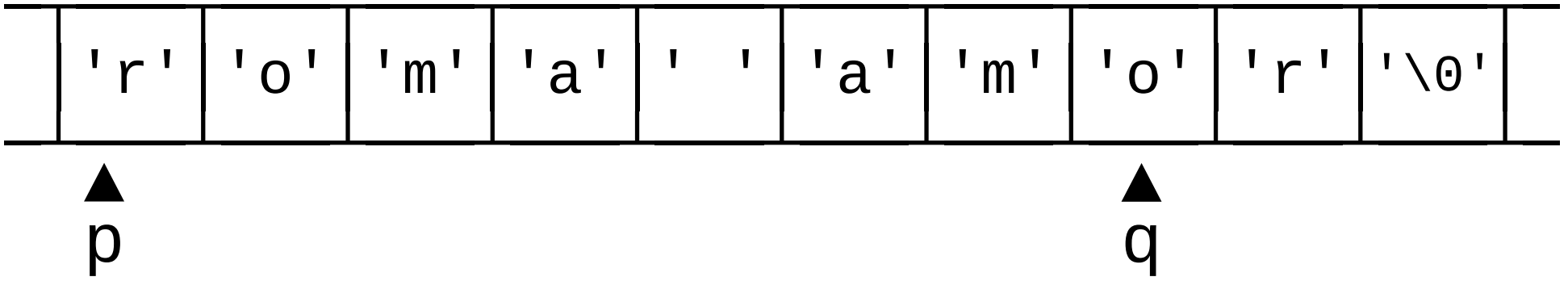

```
int palindromo(const char *p) {
  const char *q = p;

  while (*q != '\0') ++q;
  --q;

  while (p < q) {
    if (*p != *q) return 0;
    ++p; --q;
  }
  return 1;
}
```



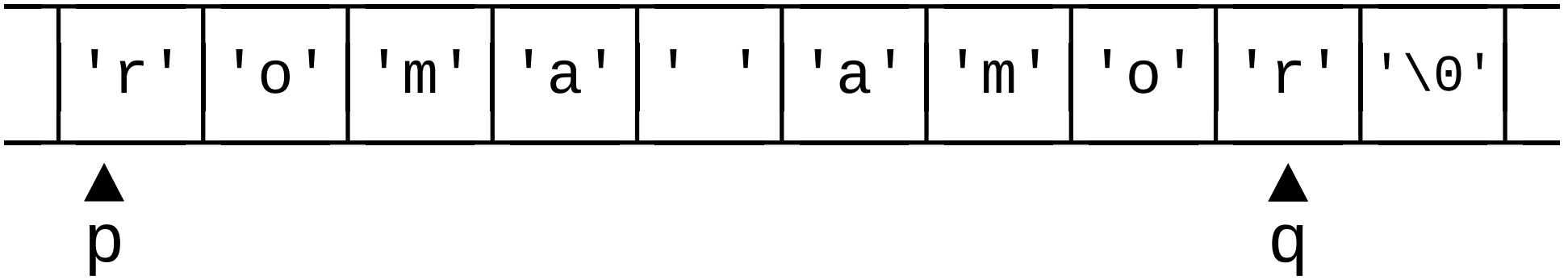

```
int palindromo(const char *p) {
  const char *q = p;

  while (*q != '\0') ++q;
  --q;

  while (p < q) {
    if (*p != *q) return 0;
    ++p; --q;
  }
  return 1;
}
```



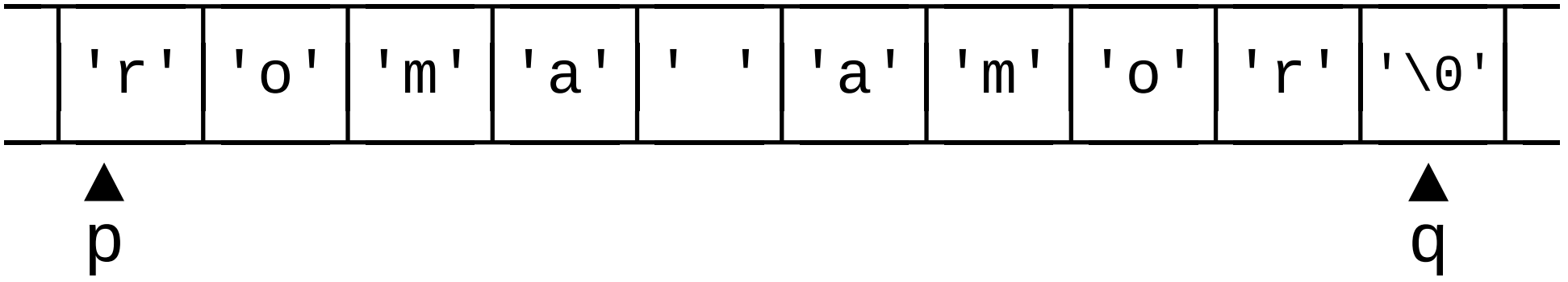

```
int palindromo(const char *p) {
  const char *q = p;

  while (*q != '\0') ++q;
  --q;

  while (p < q) {
    if (*p != *q) return 0;
    ++p; --q;
  }
  return 1;
}
```



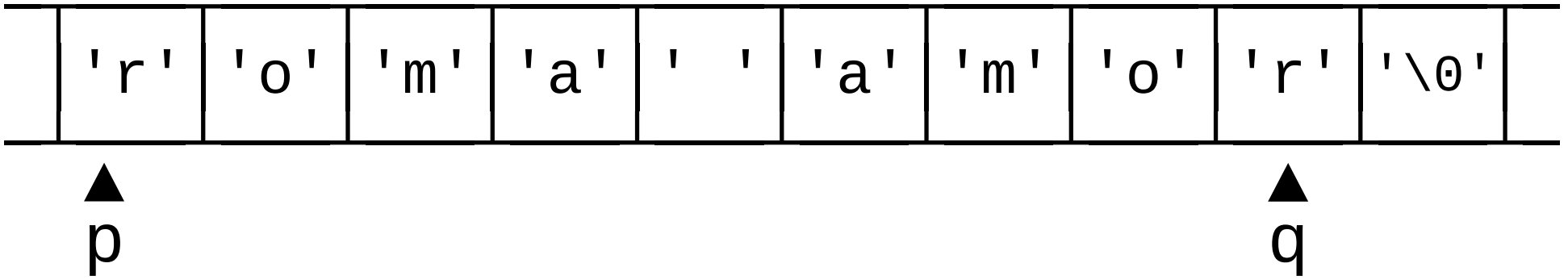

```
int palindromo(const char *p) {
  const char *q = p;

  while (*q != '\0') ++q;
  --q;

  while (p < q) {
    if (*p != *q) return 0;
    ++p; --q;
  }
  return 1;
}
```



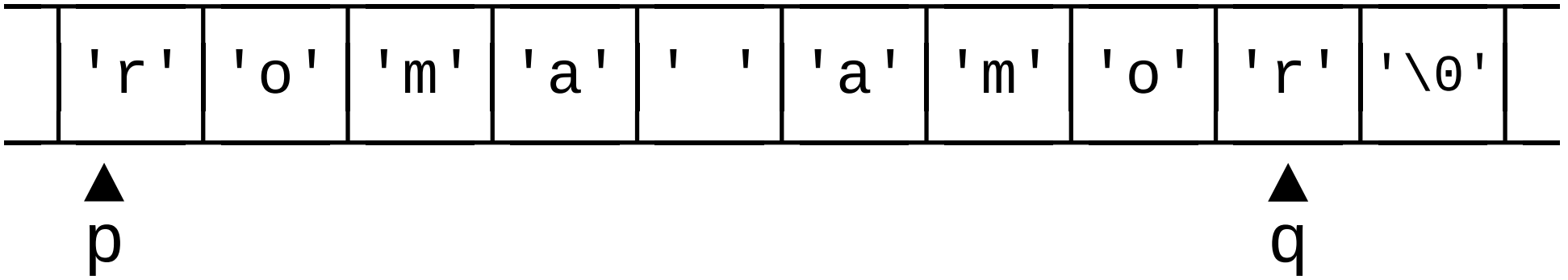

```
int palindromo(const char *p) {
  const char *q = p;

  while (*q != '\0') ++q;
  --q;

  while (p < q) {
    if (*p != *q) return 0;
    ++p; --q;
  }
  return 1;
}
```



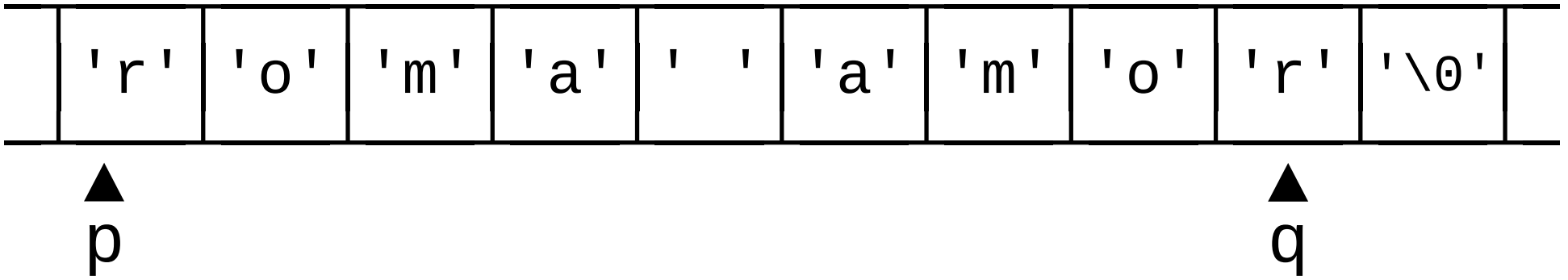

```
int palindromo(const char *p) {
  const char *q = p;

  while (*q != '\0') ++q;
  --q;

  while (p < q) {
    if (*p != *q) return 0;
    ++p; --q;
  }
  return 1;
}
```



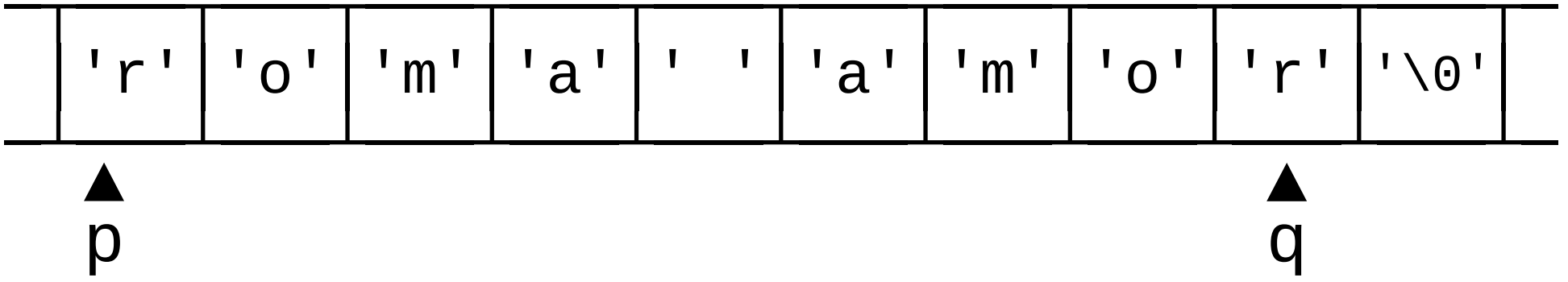

```
int palindromo(const char *p) {
  const char *q = p;

  while (*q != '\0') ++q;
  --q;

  while (p < q) {
    if (*p != *q) return 0;
    ++p; --q;
  }
  return 1;
}
```



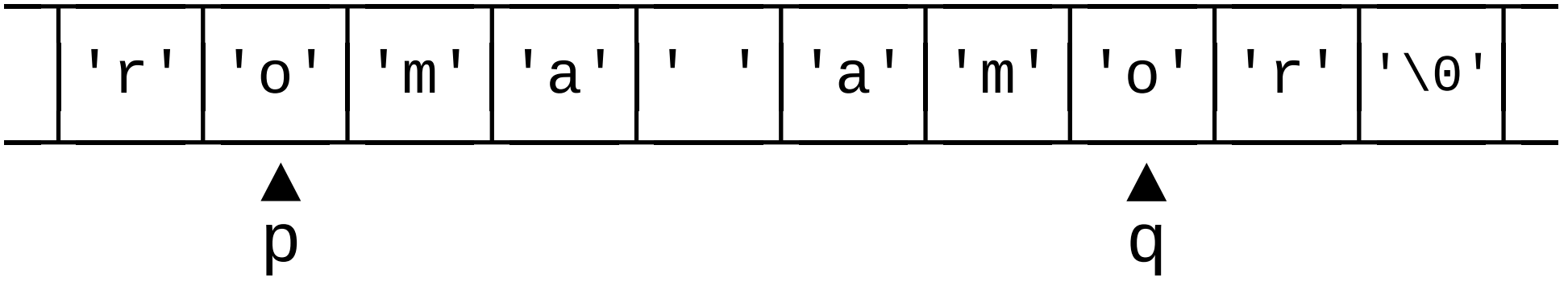

```
int palindromo(const char *p) {
  const char *q = p;

  while (*q != '\0') ++q;
  --q;

  while (p < q) {
    if (*p != *q) return 0;
    ++p; --q;
  }
  return 1;
}
```



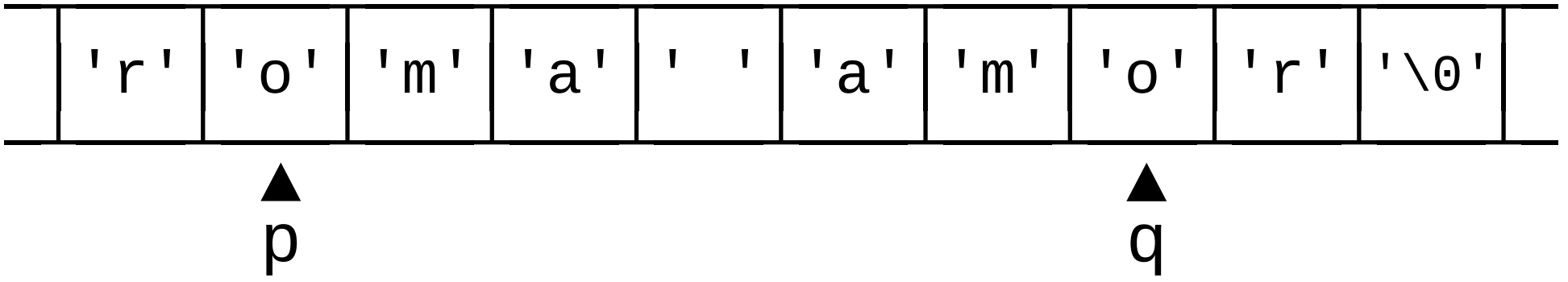

```
int palindromo(const char *p) {
  const char *q = p;

  while (*q != '\0') ++q;
  --q;

  while (p < q) {
  ▶ if (*p != *q) return 0;
    ++p; --q;
  }
  return 1;
}
```



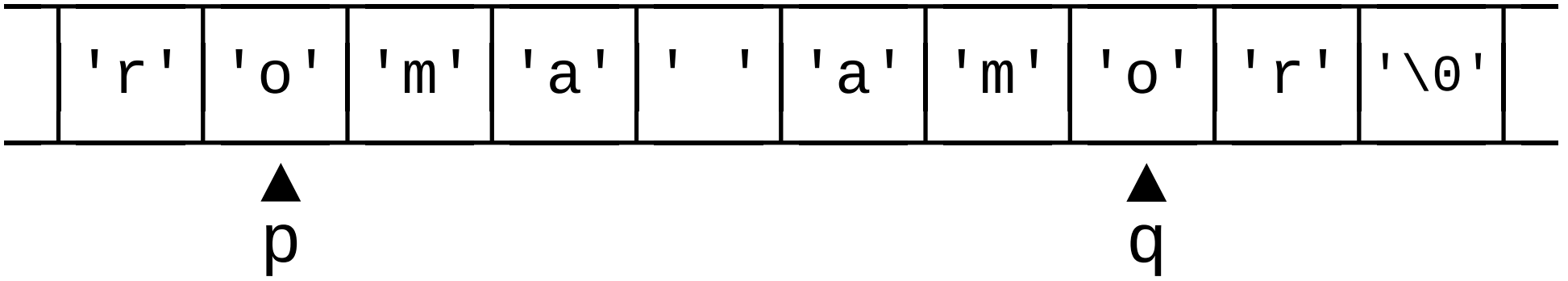

```
int palindromo(const char *p) {
  const char *q = p;

  while (*q != '\0') ++q;
  --q;

  while (p < q) {
    if (*p != *q) return 0;
    ++p; --q;
  }
  return 1;
}
```



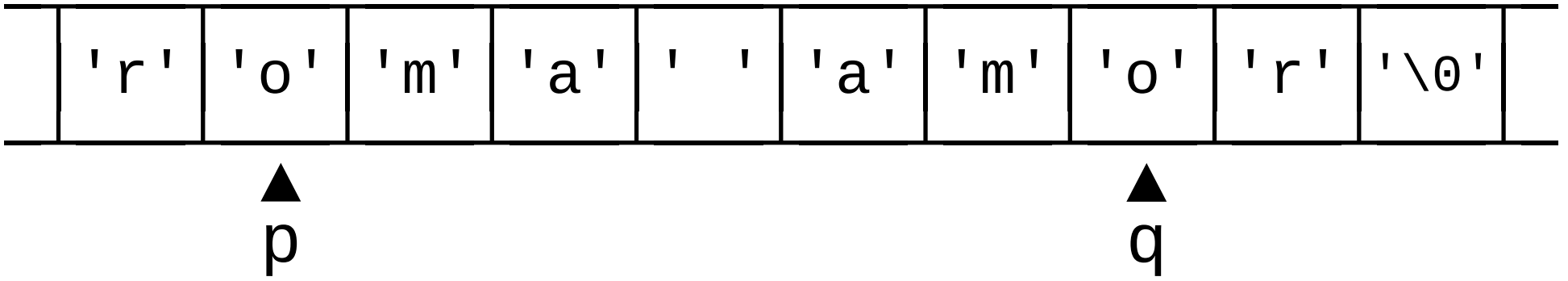

```
int palindromo(const char *p) {
  const char *q = p;

  while (*q != '\0') ++q;
  --q;

  while (p < q) {
    if (*p != *q) return 0;
    ++p; --q;
  }
  return 1;
}
```



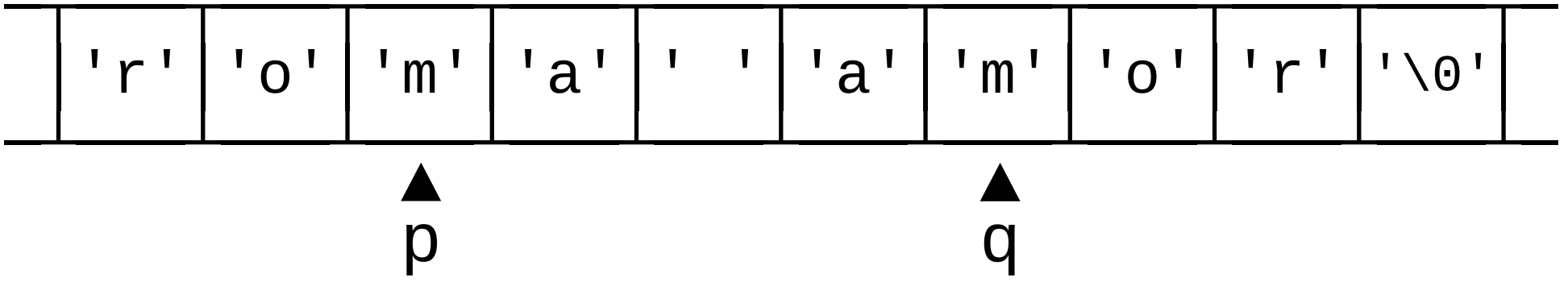

```
int palindromo(const char *p) {
  const char *q = p;

  while (*q != '\0') ++q;
  --q;

  while (p < q) {
    if (*p != *q) return 0;
    ++p; --q;
  }
  return 1;
}
```



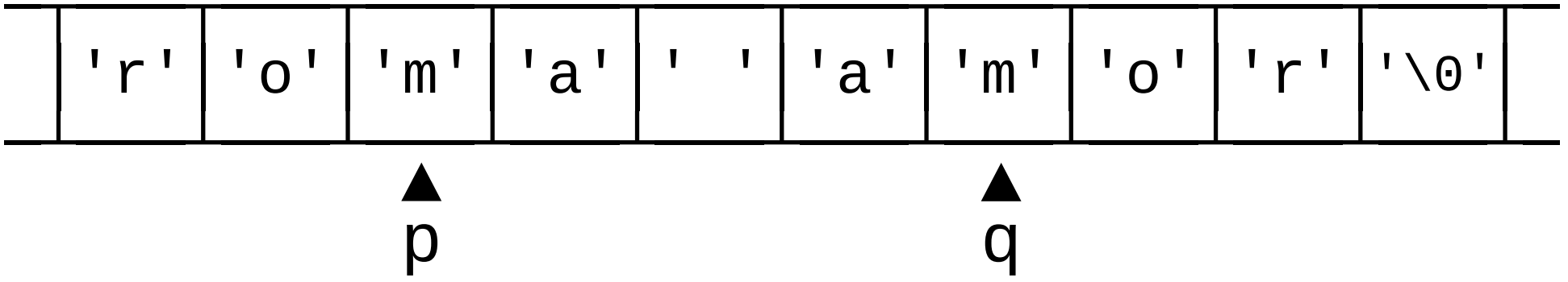

```
int palindromo(const char *p) {
  const char *q = p;

  while (*q != '\0') ++q;
  --q;

  while (p < q) {
  ▶ if (*p != *q) return 0;
    ++p; --q;
  }
  return 1;
}
```



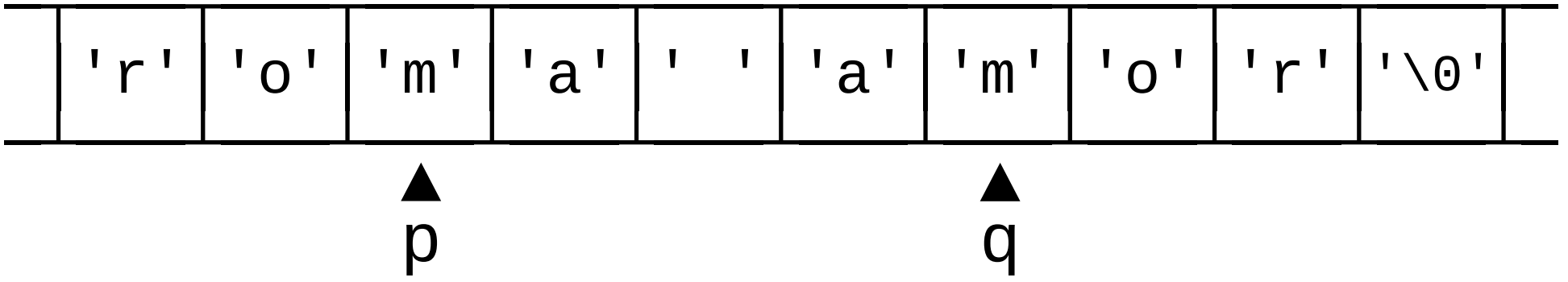

```
int palindromo(const char *p) {
  const char *q = p;

  while (*q != '\0') ++q;
  --q;

  while (p < q) {
    if (*p != *q) return 0;
    ++p; --q;
  }
  return 1;
}
```



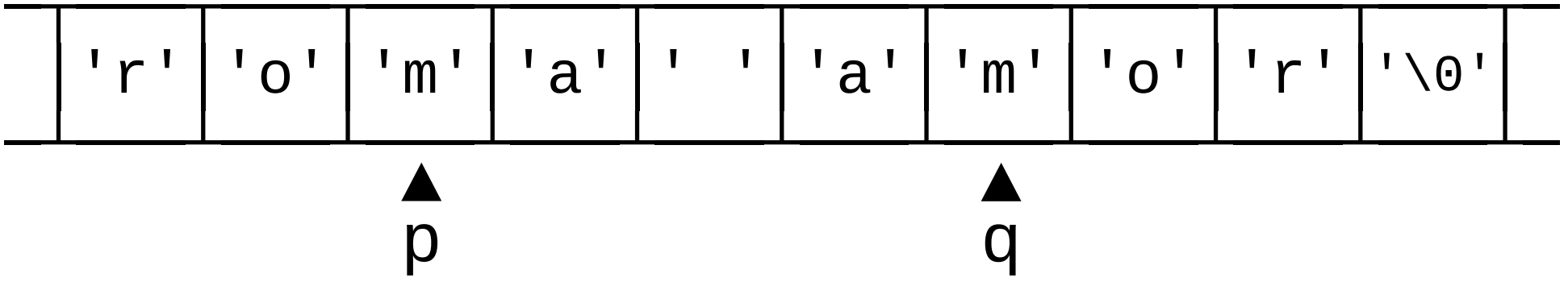

```
int palindromo(const char *p) {
  const char *q = p;

  while (*q != '\0') ++q;
  --q;

  while (p < q) {
    if (*p != *q) return 0;
    ++p; --q;
  }
  return 1;
}
```



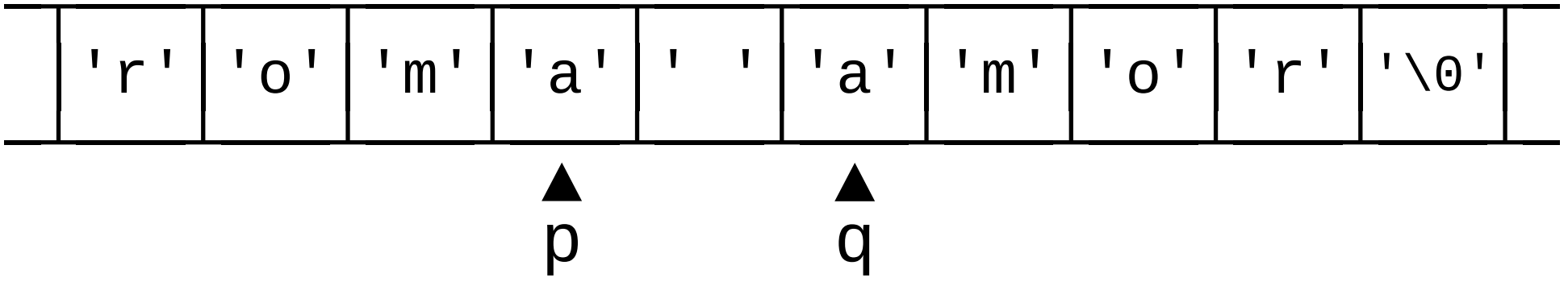

```
int palindromo(const char *p) {
  const char *q = p;

  while (*q != '\0') ++q;
  --q;

  while (p < q) {
    if (*p != *q) return 0;
    ++p; --q;
  }
  return 1;
}
```



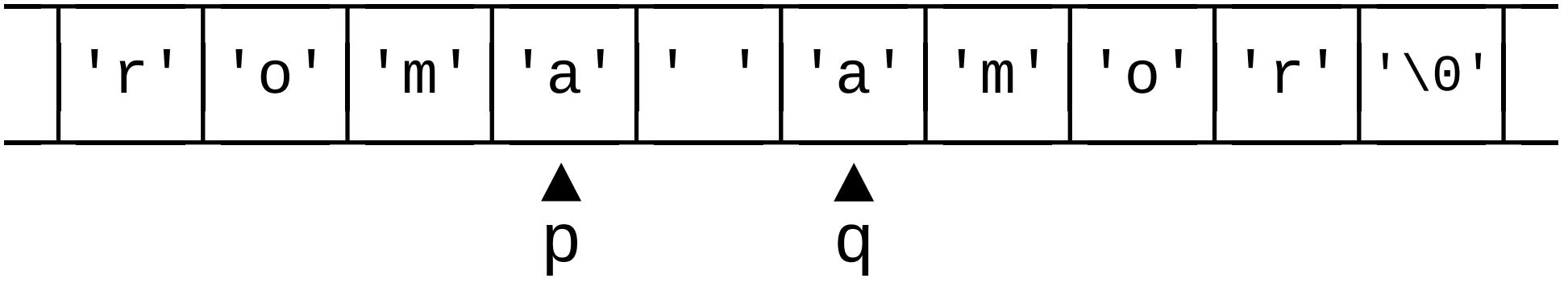

```
int palindromo(const char *p) {
  const char *q = p;

  while (*q != '\0') ++q;
  --q;

  while (p < q) {
  ▶ if (*p != *q) return 0;
    ++p; --q;
  }
  return 1;
}
```



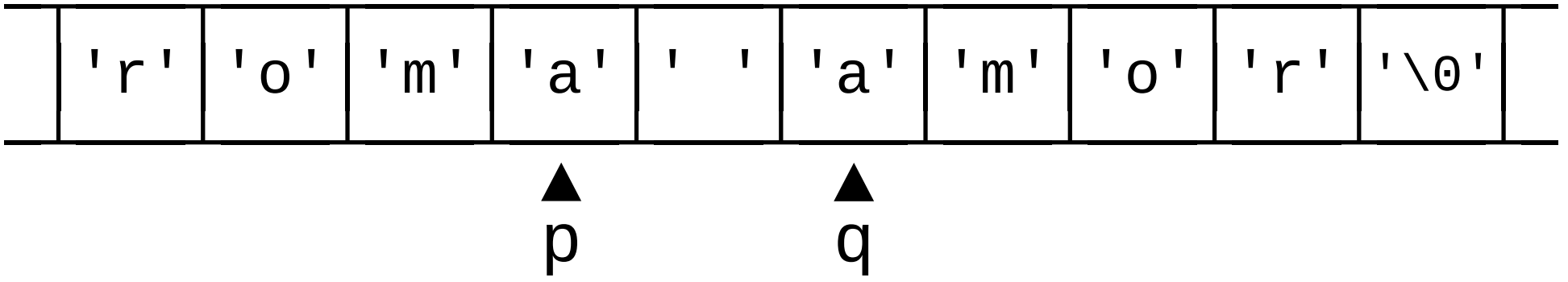

```
int palindromo(const char *p) {
  const char *q = p;

  while (*q != '\0') ++q;
  --q;

  while (p < q) {
    if (*p != *q) return 0;
    ++p; --q;
  }
  return 1;
}
```



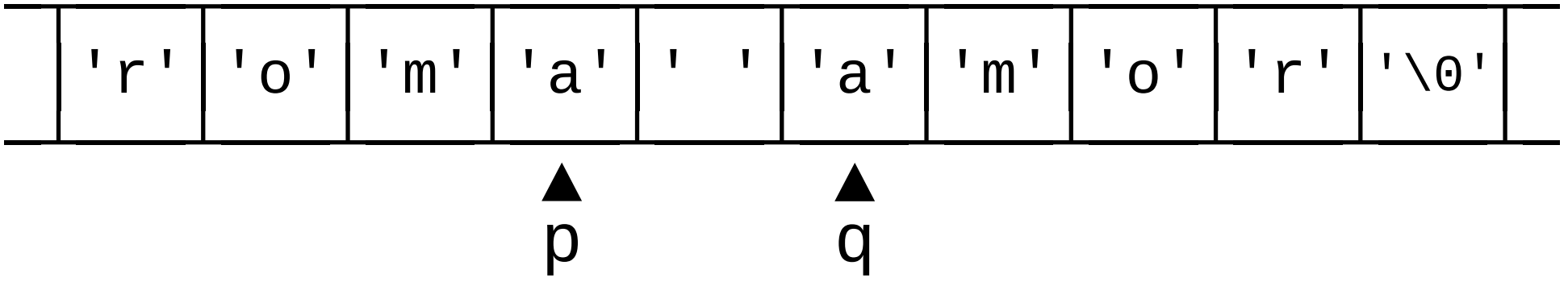

```
int palindromo(const char *p) {
  const char *q = p;

  while (*q != '\0') ++q;
  --q;

  while (p < q) {
    if (*p != *q) return 0;
    ++p; --q;
  }
  return 1;
}
```



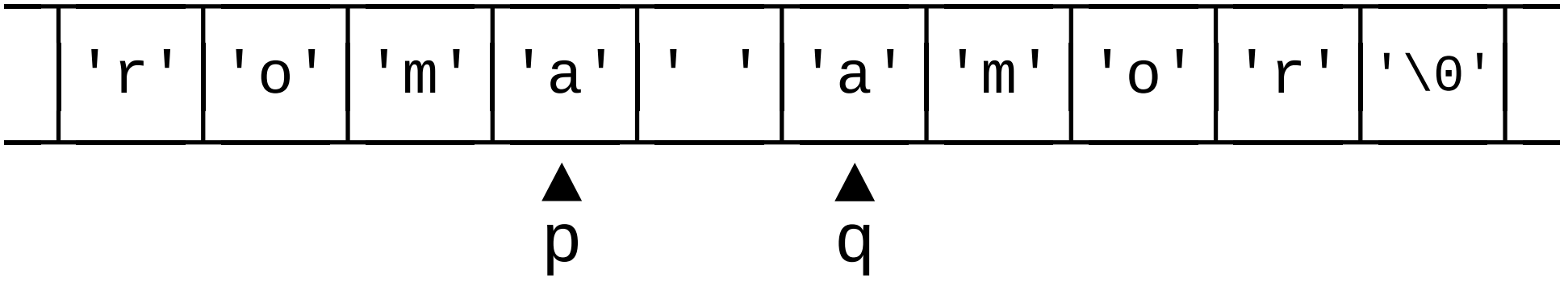

```
int palindromo(const char *p) {
  const char *q = p;

  while (*q != '\0') ++q;
  --q;

  while (p < q) {
    if (*p != *q) return 0;
    ++p; --q;
  }
  return 1;
}
```



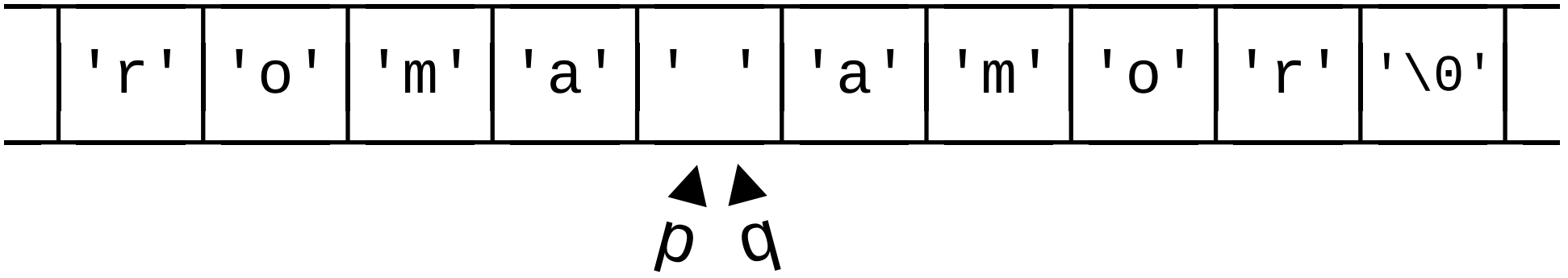

```
int palindromo(const char *p) {
  const char *q = p;

  while (*q != '\0') ++q;
  --q;

  while (p < q) {
  ▶ if (*p != *q) return 0;
    ++p; --q;
  }
  return 1;
}
```



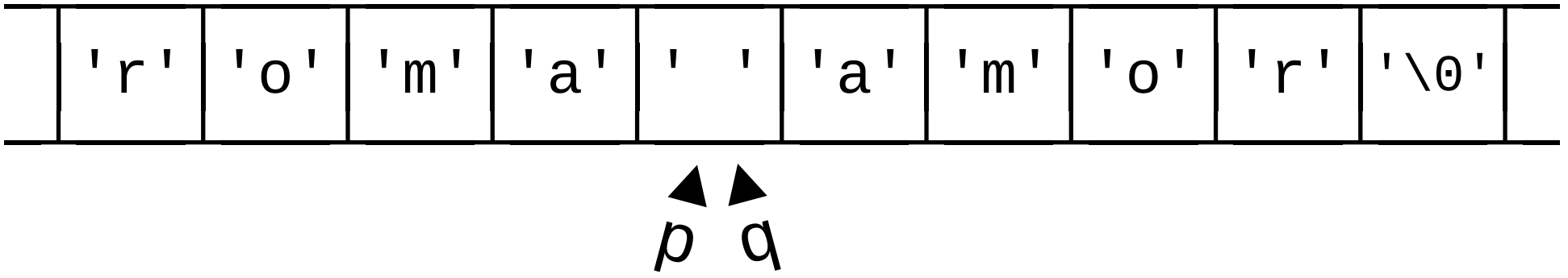

```
int palindromo(const char *p) {
  const char *q = p;

  while (*q != '\0') ++q;
  --q;

  while (p < q) {
    if (*p != *q) return 0;
    ++p; --q;
  }
  return 1;
}
```



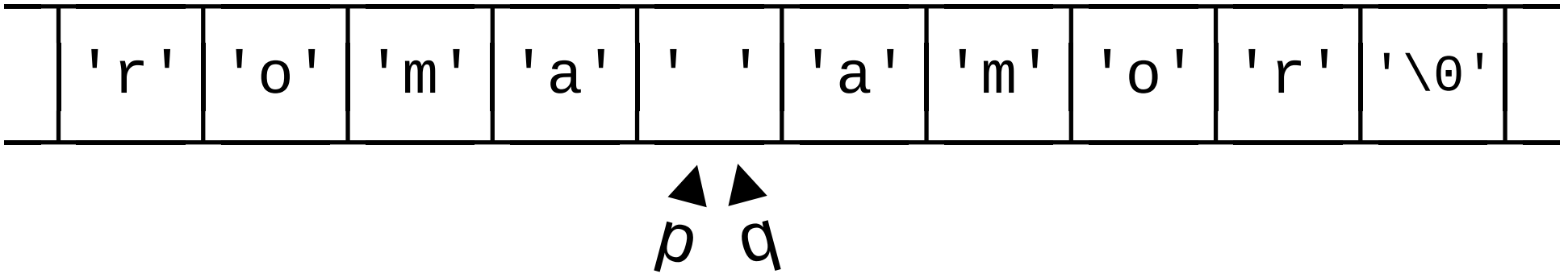

```
int palindromo(const char *p) {
  const char *q = p;

  while (*q != '\0') ++q;
  --q;

  while (p < q) {
    if (*p != *q) return 0;
    ++p; --q;
  }
  return 1;
}
```



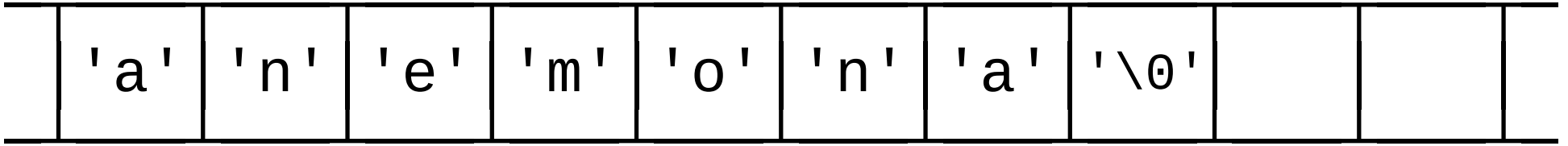

```
int palindromo(const char *p) {
  const char *q = p;

  while (*q != '\0') ++q;
  --q;

  while (p < q) {
    if (*p != *q) return 0;
    ++p; --q;
  }
  return 1;
}
```



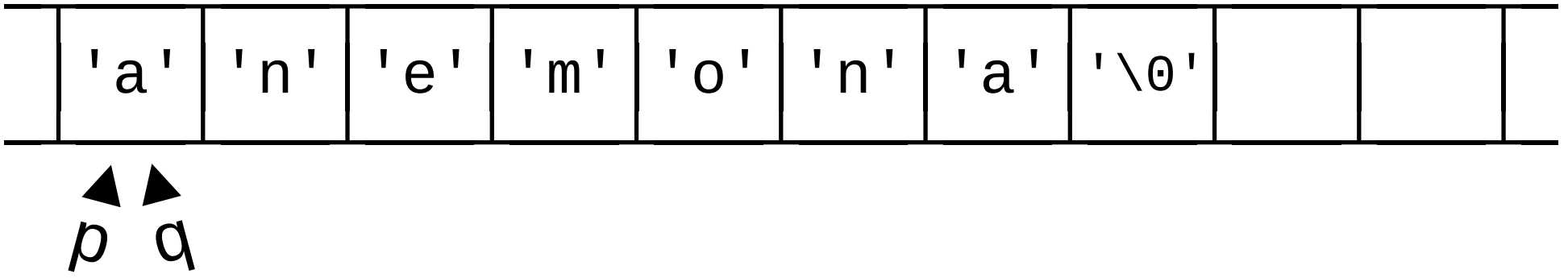

```
int palindromo(const char *p) {
  const char *q = p;

  while (*q != '\0') ++q;
  --q;

  while (p < q) {
    if (*p != *q) return 0;
    ++p; --q;
  }
  return 1;
}
```



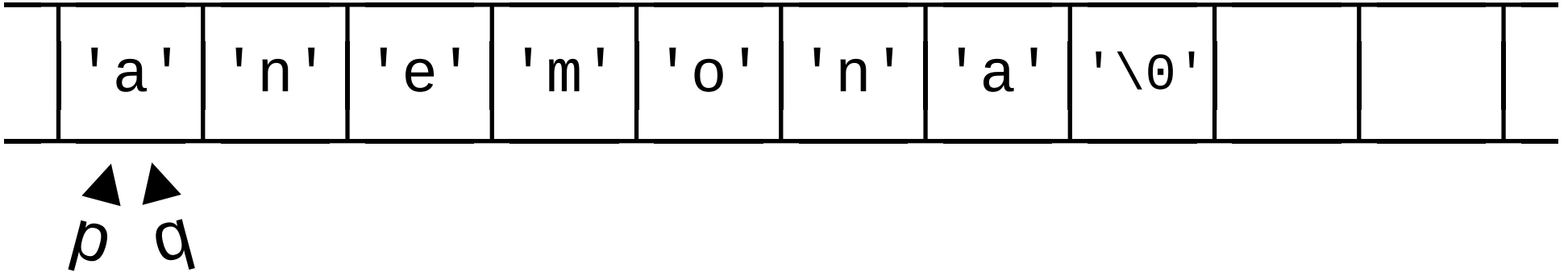

```
int palindromo(const char *p) {
  const char *q = p;

  while (*q != '\0') ++q;
  --q;

  while (p < q) {
    if (*p != *q) return 0;
    ++p; --q;
  }
  return 1;
}
```



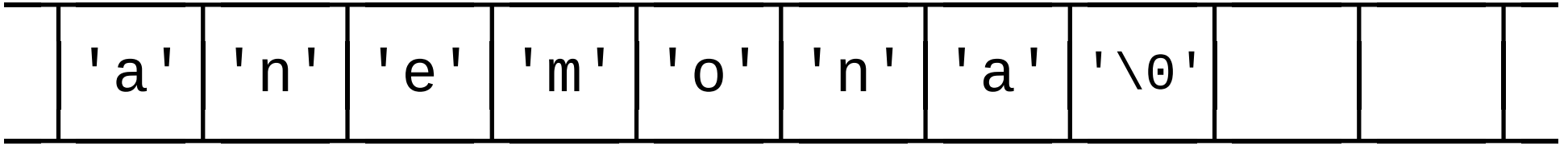

```
int palindromo(const char *p) {
  const char *q = p;

  while (*q != '\0') ++q;
  --q;

  while (p < q) {
    if (*p != *q) return 0;
    ++p; --q;
  }
  return 1;
}
```



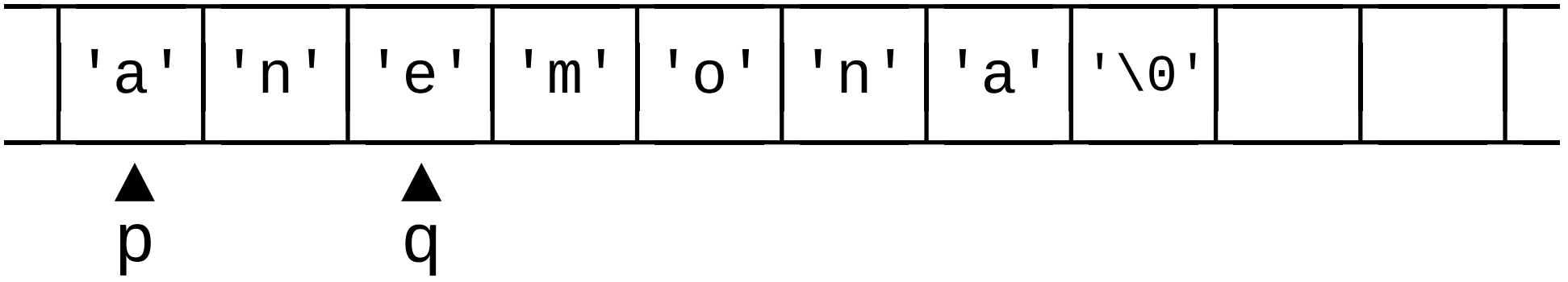

```
int palindromo(const char *p) {
  const char *q = p;

  while (*q != '\0') ++q;
  --q;

  while (p < q) {
    if (*p != *q) return 0;
    ++p; --q;
  }
  return 1;
}
```



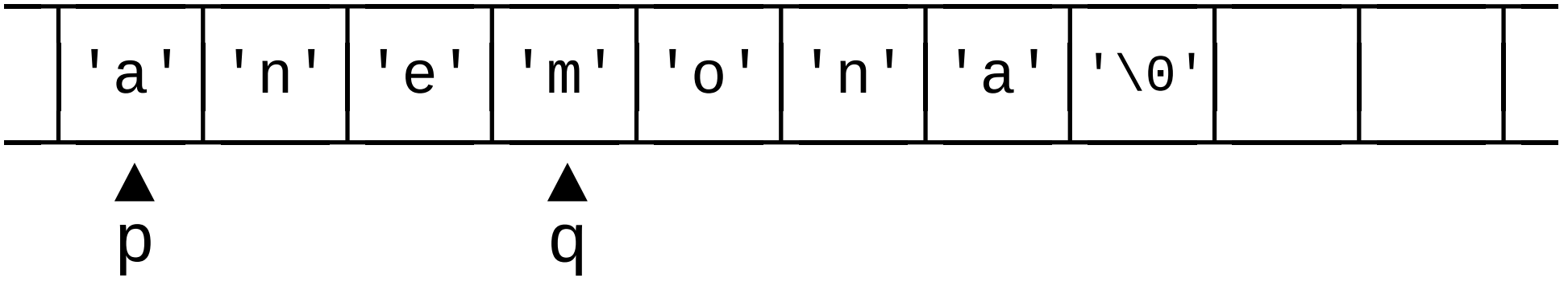

```
int palindromo(const char *p) {
  const char *q = p;

  while (*q != '\0') ++q;
  --q;

  while (p < q) {
    if (*p != *q) return 0;
    ++p; --q;
  }
  return 1;
}
```



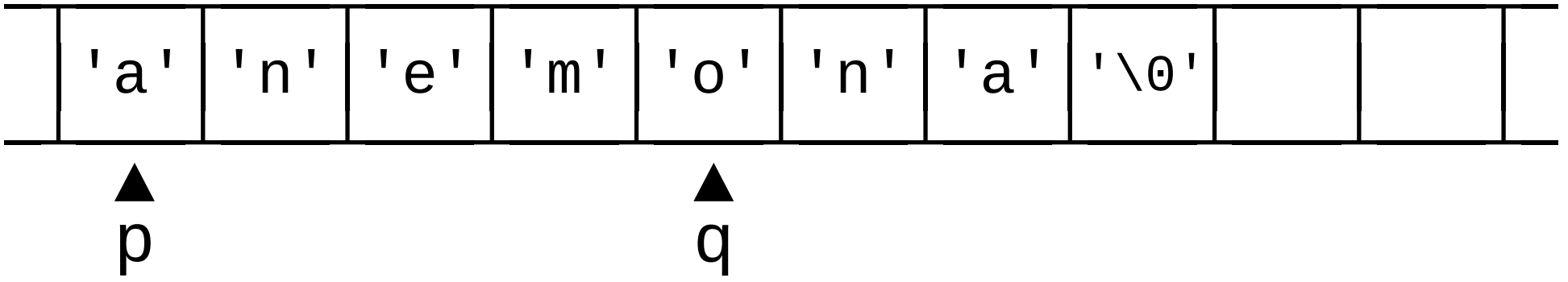

```
int palindromo(const char *p) {
  const char *q = p;

  while (*q != '\0') ++q;
  --q;

  while (p < q) {
    if (*p != *q) return 0;
    ++p; --q;
  }
  return 1;
}
```



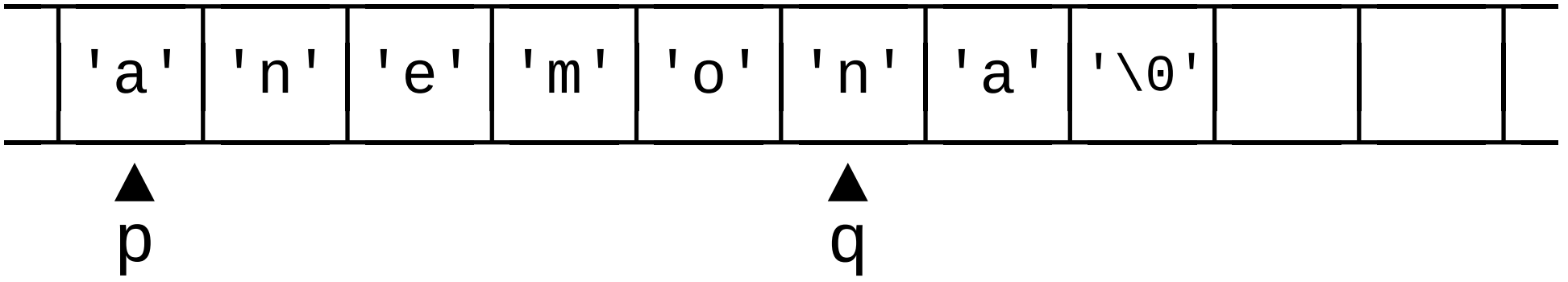

```
int palindromo(const char *p) {
  const char *q = p;

  while (*q != '\0') ++q;
  --q;

  while (p < q) {
    if (*p != *q) return 0;
    ++p; --q;
  }
  return 1;
}
```



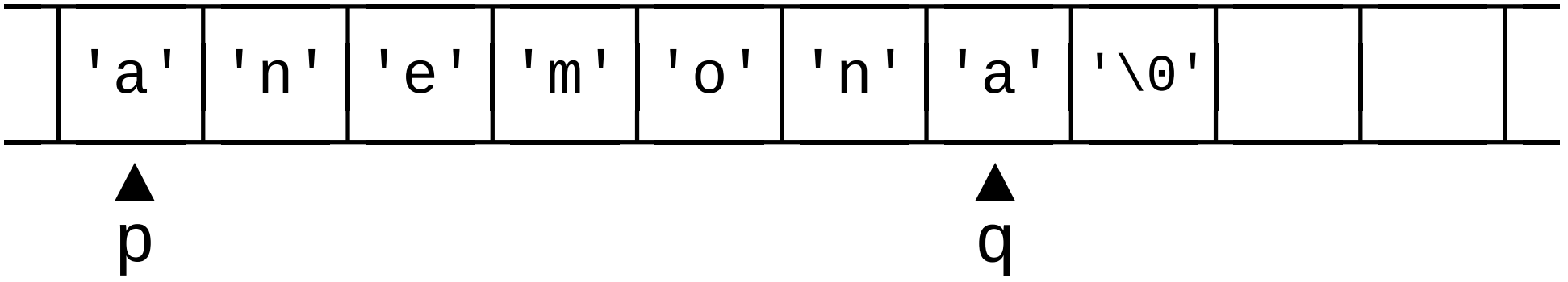

```
int palindromo(const char *p) {
  const char *q = p;

  while (*q != '\0') ++q;
  --q;

  while (p < q) {
    if (*p != *q) return 0;
    ++p; --q;
  }
  return 1;
}
```



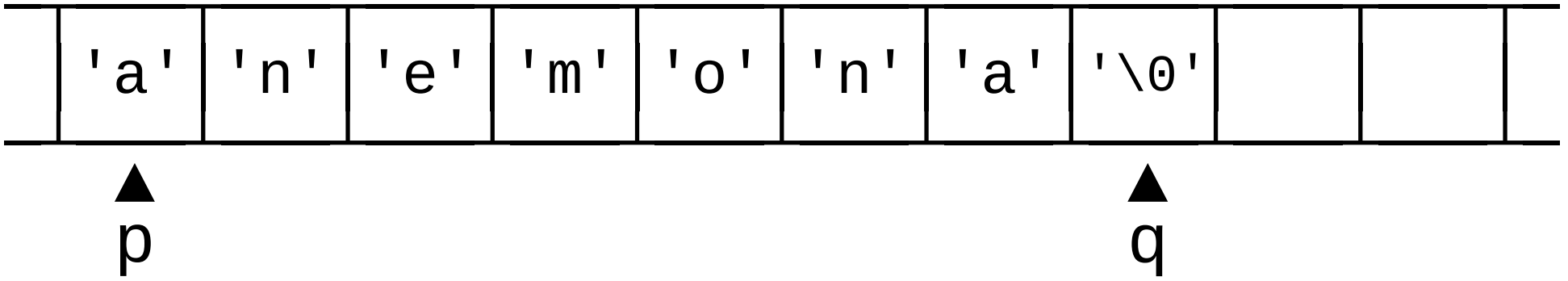

```
int palindromo(const char *p) {
  const char *q = p;

  while (*q != '\0') ++q;
  --q;

  while (p < q) {
    if (*p != *q) return 0;
    ++p; --q;
  }
  return 1;
}
```



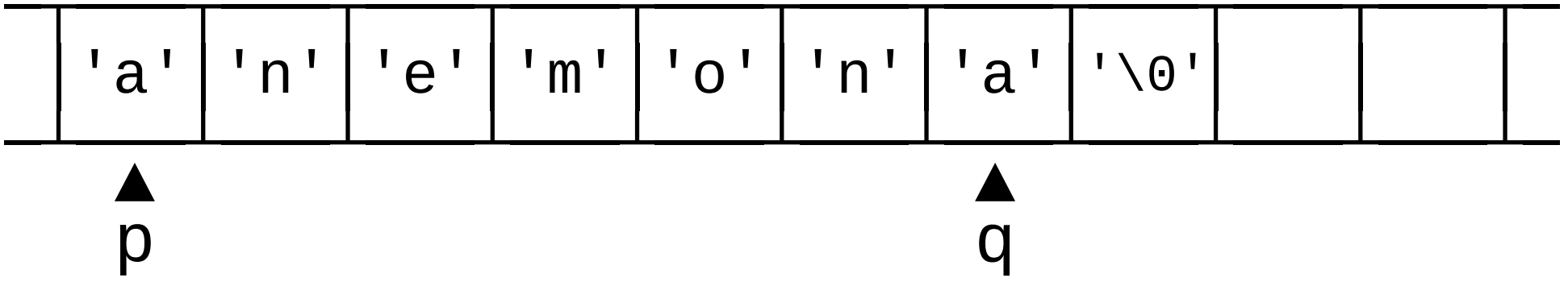

```
int palindromo(const char *p) {
  const char *q = p;

  while (*q != '\0') ++q;
  --q;

  while (p < q) {
    if (*p != *q) return 0;
    ++p; --q;
  }
  return 1;
}
```



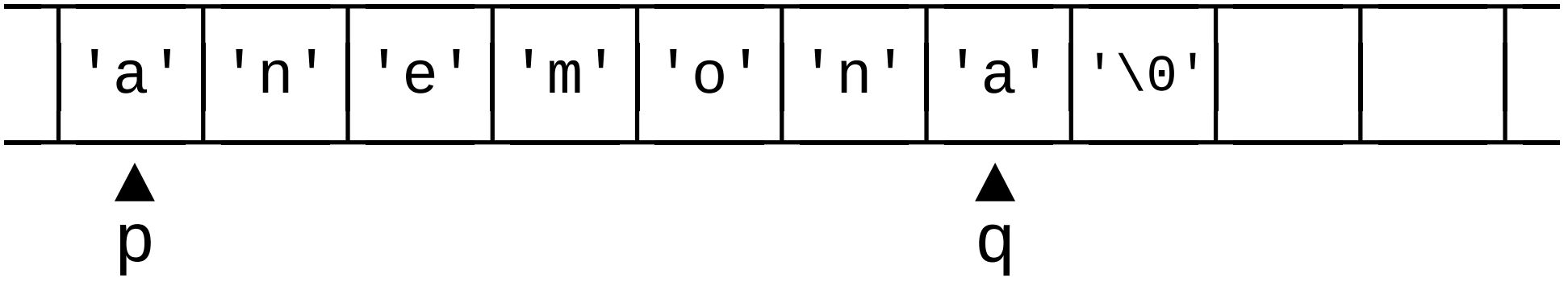

```
int palindromo(const char *p) {
  const char *q = p;

  while (*q != '\0') ++q;
  --q;

  while (p < q) {
    if (*p != *q) return 0;
    ++p; --q;
  }
  return 1;
}
```



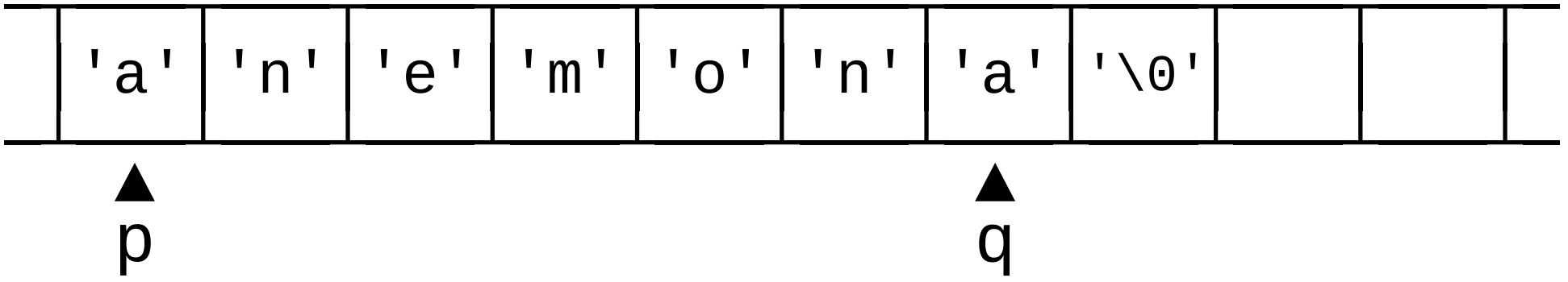

```
int palindromo(const char *p) {
  const char *q = p;

  while (*q != '\0') ++q;
  --q;

  while (p < q) {
    if (*p != *q) return 0;
    ++p; --q;
  }
  return 1;
}
```



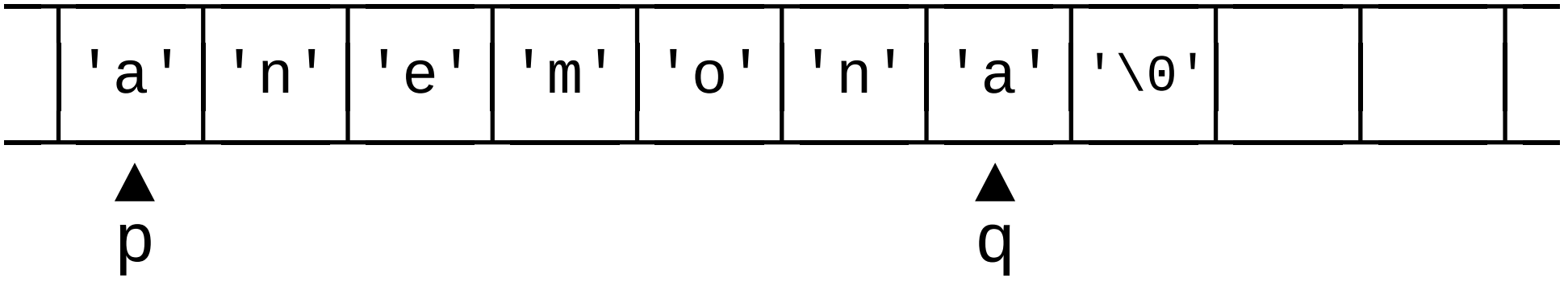

```
int palindromo(const char *p) {
  const char *q = p;

  while (*q != '\0') ++q;
  --q;

  while (p < q) {
    if (*p != *q) return 0;
    ++p; --q;
  }
  return 1;
}
```



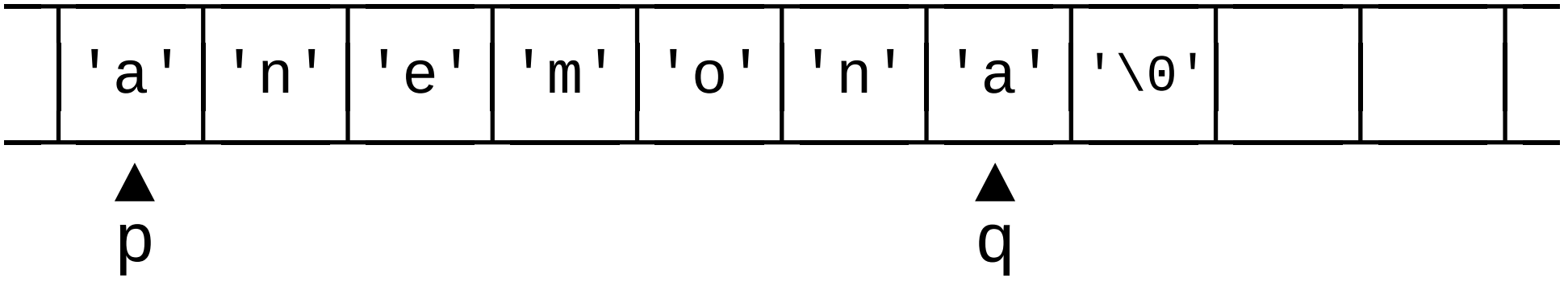

```
int palindromo(const char *p) {
  const char *q = p;

  while (*q != '\0') ++q;
  --q;

▶ while (p < q) {
    if (*p != *q) return 0;
    ++p; --q;
  }
  return 1;
}
```



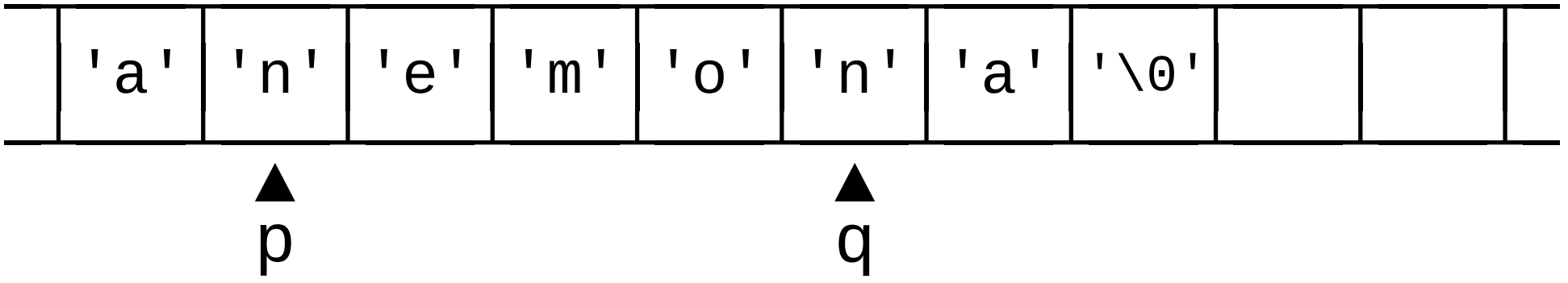

```
int palindromo(const char *p) {
  const char *q = p;

  while (*q != '\0') ++q;
  --q;

  while (p < q) {
  ▶ if (*p != *q) return 0;
    ++p; --q;
  }
  return 1;
}
```



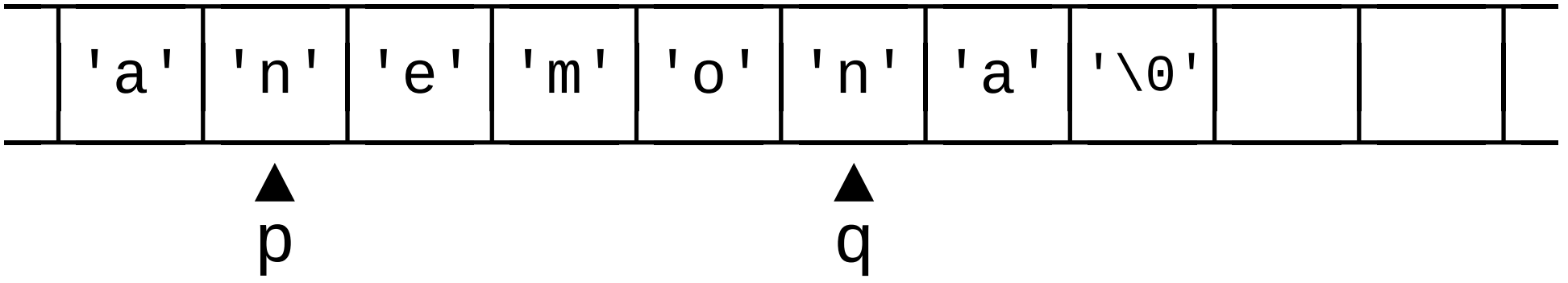

```
int palindromo(const char *p) {
  const char *q = p;

  while (*q != '\0') ++q;
  --q;

  while (p < q) {
  ▶ if (*p != *q) return 0;
    ++p; --q;
  }
  return 1;
}
```



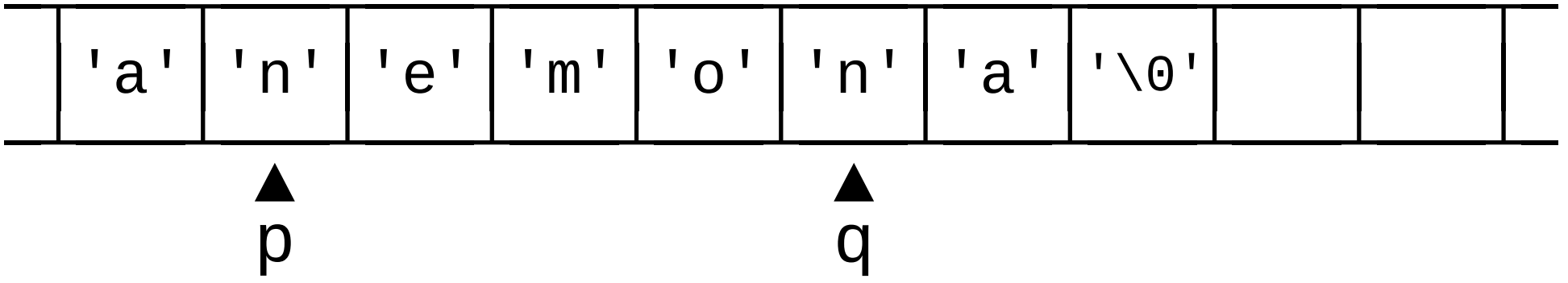

```
int palindromo(const char *p) {
  const char *q = p;

  while (*q != '\0') ++q;
  --q;

  while (p < q) {
    if (*p != *q) return 0;
    ++p; --q;
  }
  return 1;
}
```



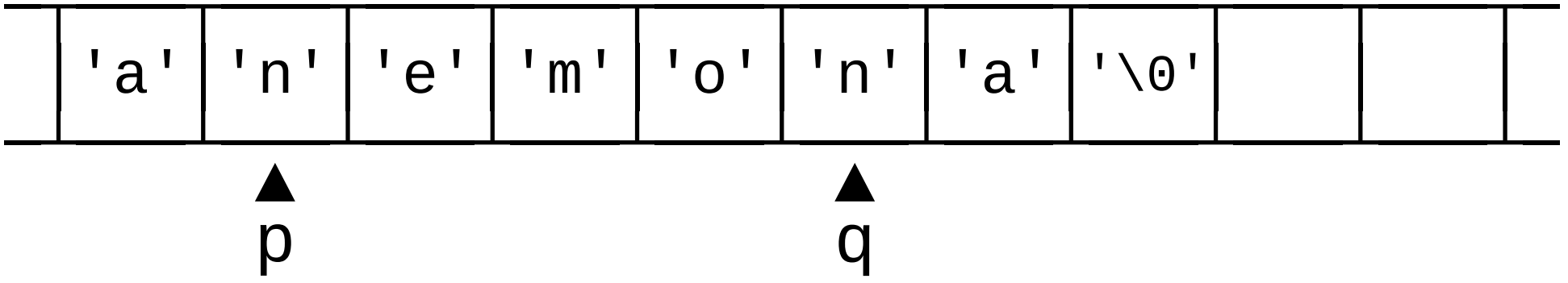

```
int palindromo(const char *p) {
  const char *q = p;

  while (*q != '\0') ++q;
  --q;

  while (p < q) {
    if (*p != *q) return 0;
    ++p; --q;
  }
  return 1;
}
```



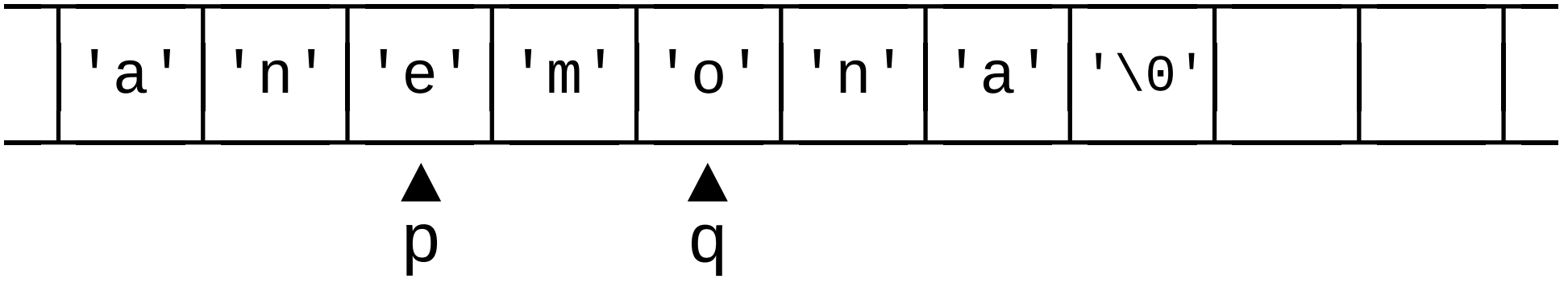

```
int palindromo(const char *p) {
  const char *q = p;

  while (*q != '\0') ++q;
  --q;

▶ while (p < q) {
    if (*p != *q) return 0;
    ++p; --q;
  }
  return 1;
}
```



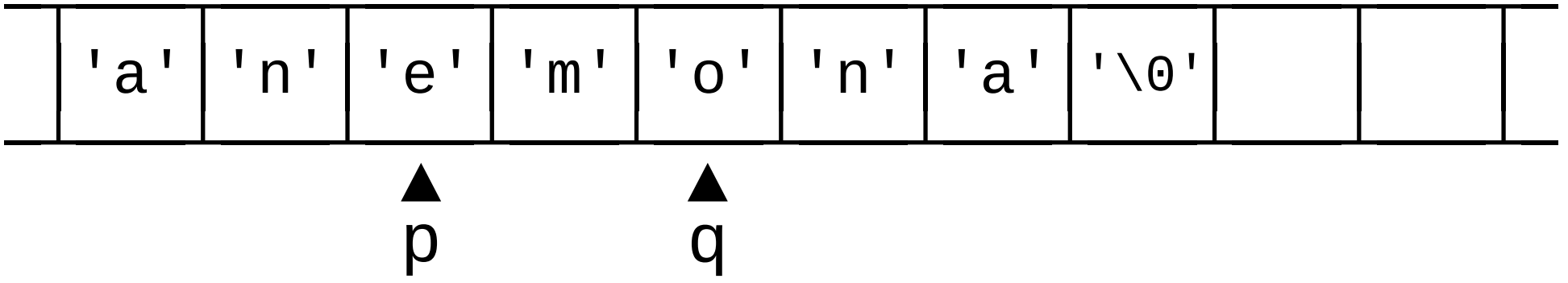

```
int palindromo(const char *p) {
  const char *q = p;

  while (*q != '\0') ++q;
  --q;

  while (p < q) {
  ▶ if (*p != *q) return 0;
    ++p; --q;
  }
  return 1;
}
```



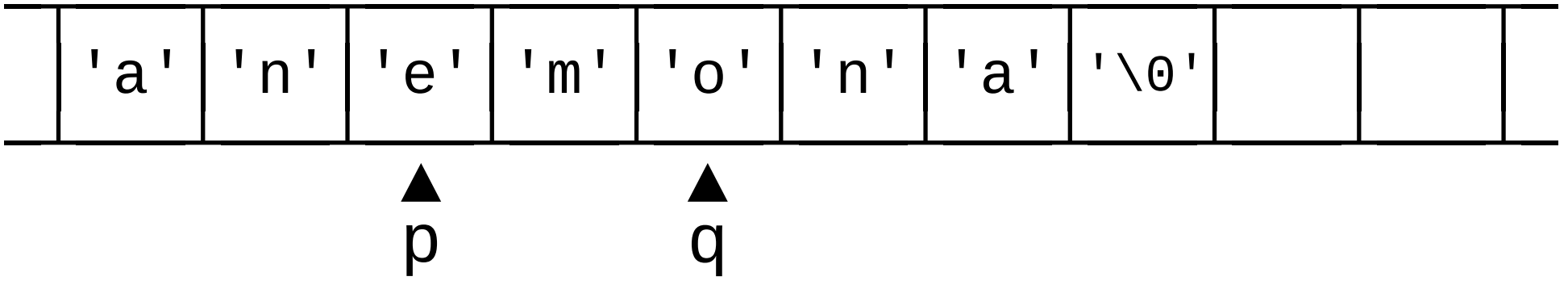

```
int palindromo(const char *p) {
  const char *q = p;

  while (*q != '\0') ++q;
  --q;

  while (p < q) {
    if (*p != *q) return 0;
    ++p; --q;
  }
  return 1;
}
```



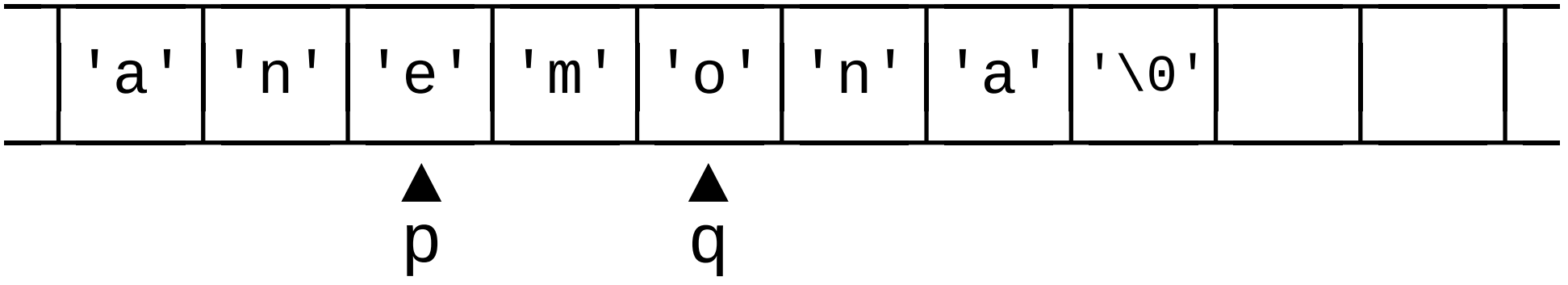

```
int palindromo(const char *p) {
  const char *q = p;

  while (*q != '\0') ++q;
  --q;

  while (p < q) {
    if (*p != *q) return 0;
    ++p; --q;
  }
  return 1;
}
```




```
int palindromo(const char *p) {
  const char *q = p;

  while (*q != '\0') ++q;
  --q;

  while (p < q) {
    if (*p != *q) return 0;
    ++p; --q;
  }
  return 1;
}
```

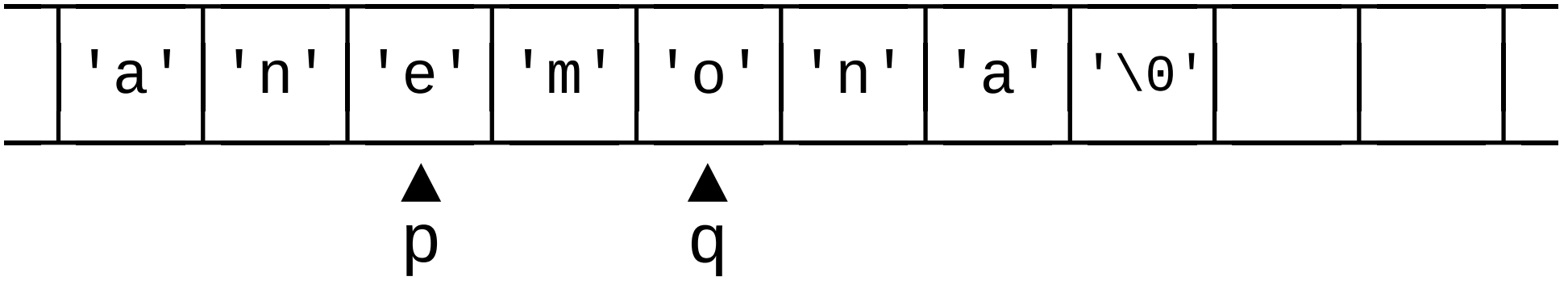



```
int palindromo(const char *p) {
  const char *q = p;

  while (*q != '\0') ++q;
  --q;

  while (p < q) {
  ▶ if (*p != *q) return 0;
    ++p; --q;
  }
  return 1;
}
```

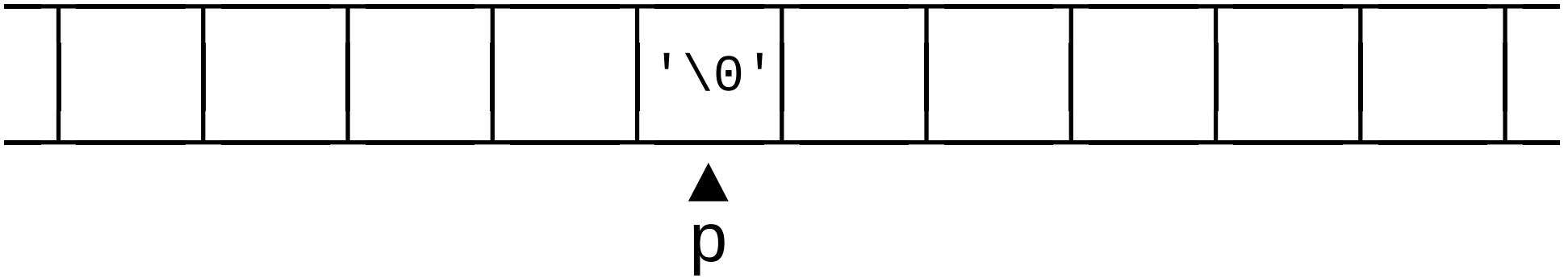



```
int palindromo(const char *p) {
  const char *q = p;

  while (*q != '\0') ++q;
  --q;

  while (p < q) {
    if (*p != *q) return 0;
    ++p; --q;
  }
  return 1;
}
```

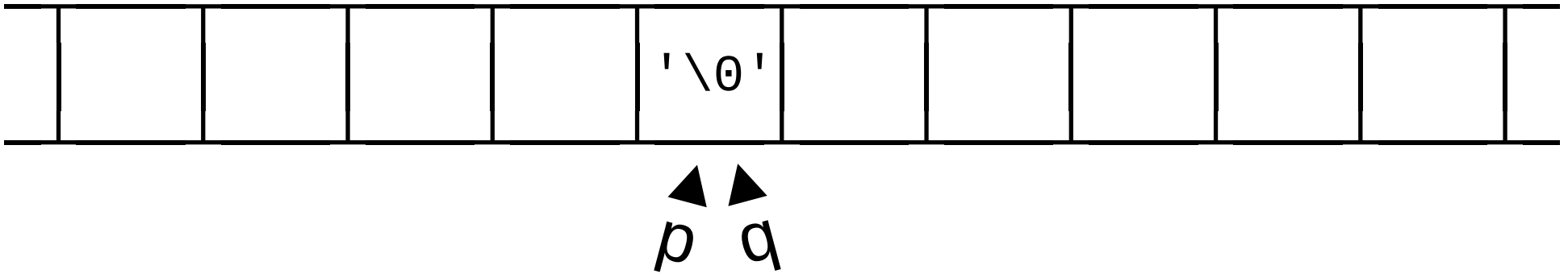
*verdadeiro, pois 'e' ≠ 'o'*



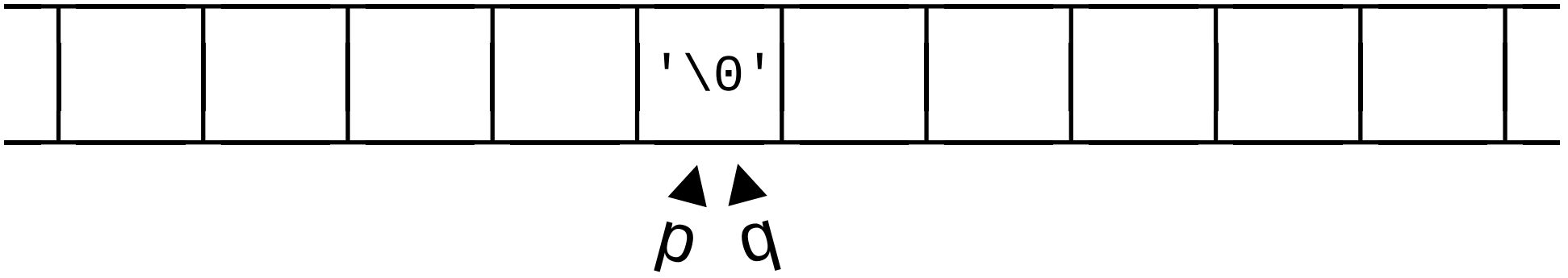

```
int palindromo(const char *p) {
  const char *q = p;

  while (*q != '\0') ++q;
  --q;

  while (p < q) {
    if (*p != *q)▶return 0;
    ++p; --q;
  }
  return 1;
}
```



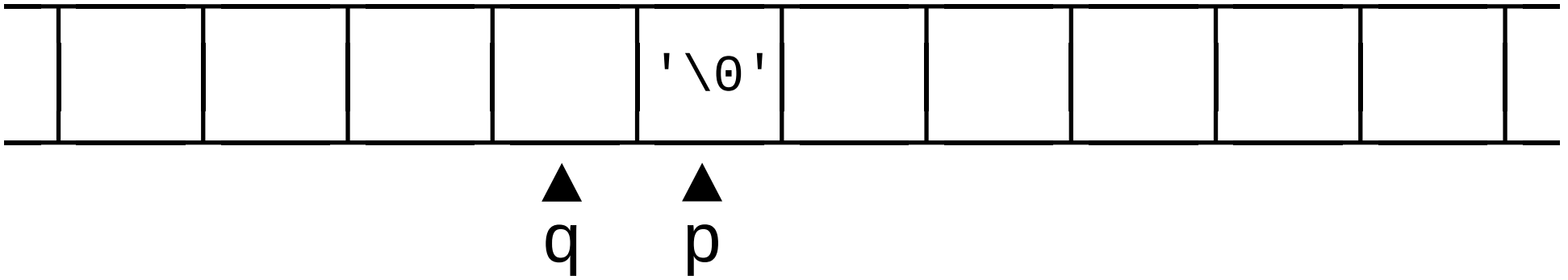

```
int palindromo(const char *p) {
  const char *q = p;

  while (*q != '\0') ++q;
  --q;

  while (p < q) {
    if (*p != *q) return 0;
    ++p; --q;
  }
  return 1;
}
```



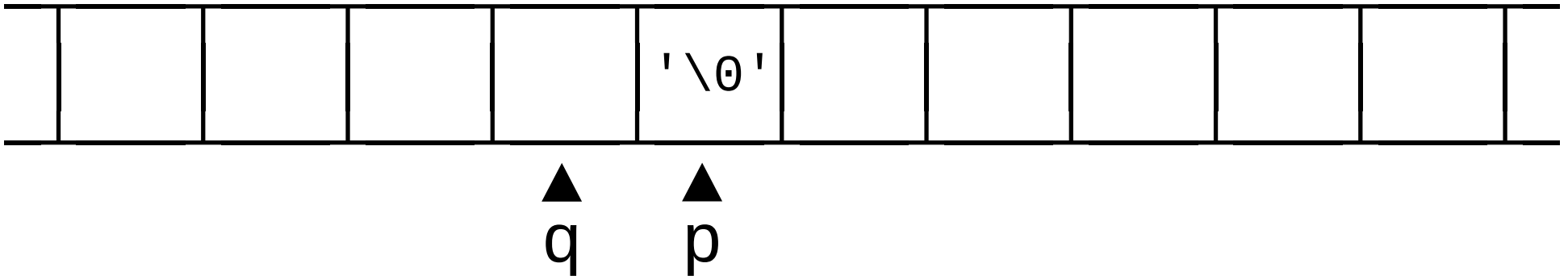

```
int palindromo(const char *p) {
  const char *q = p;

  while (*q != '\0') ++q;
  --q;

  while (p < q) {
    if (*p != *q) return 0;
    ++p; --q;
  }
  return 1;
}
```



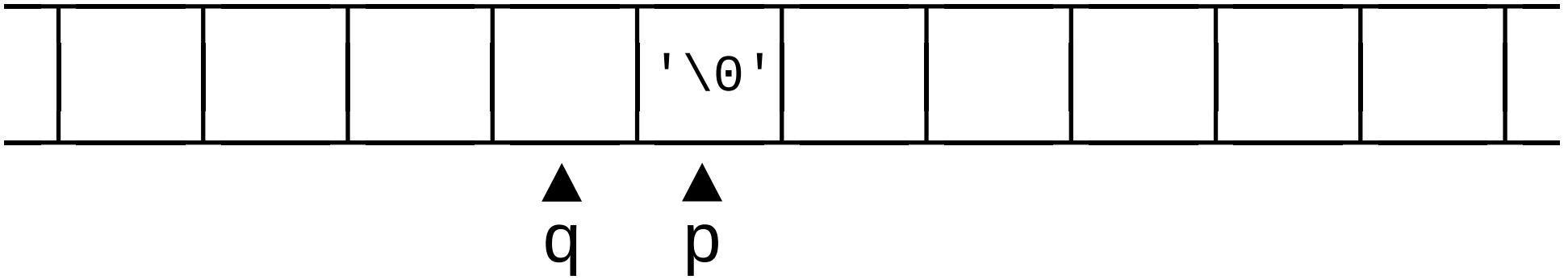

```
int palindromo(const char *p) {
  const char *q = p;

  while (*q != '\0') ++q;
  --q;

  while (p < q) {
    if (*p != *q) return 0;
    ++p; --q;
  }
  return 1;
}
```




```
int palindromo(const char *p) {
  const char *q = p;

  while (*q != '\0') ++q;
  --q;

  while (p < q) {
    if (*p != *q) return 0;
    ++p; --q;
  }
  return 1;
}
```




```
int palindromo(const char *p) {
  const char *q = p;

  while (*q != '\0') ++q;
  --q;

  while (p < q) {
    if (*p != *q) return 0;
    ++p; --q;
  }
  return 1;
}
```




```
int palindromo(const char *p) {
  const char *q = p;

  while (*q != '\0') ++q;
  --q;

  while (p < q) {
    if (*p != *q) return 0;
    ++p; --q;
  }
  return 1;
}
```

# PASSAGEM DE PARÂMETROS

Crie o arquivo incrementa.c

```
void incrementa(int i) {
  ++i;
}
```

Teste no cling:

```
[cling]$ .L incrementa.c
```

# PASSAGEM DE PARÂMETROS

Crie o arquivo incrementa.c

```
void incrementa(int i) {
  ++i;
}
```

Teste no cling:

```
[cling]$ .L incrementa.c
[cling]$ int i = 0;
```

# PASSAGEM DE PARÂMETROS

Crie o arquivo incrementa.c

```
void incrementa(int i) {
  ++i;
}
```

Teste no cling:

```
[cling]$ .L incrementa.c
[cling]$ int i = 0;
[cling]$ incrementa(i)
```

# PASSAGEM DE PARÂMETROS

Crie o arquivo incrementa.c

```
void incrementa(int i) {
  ++i;
}
```

Teste no cling:

```
[cling]$ .L incrementa.c
[cling]$ int i = 0;
[cling]$ incrementa(i)
[cling]$
```

# PASSAGEM DE PARÂMETROS

Crie o arquivo incrementa.c

```
void incrementa(int i) {
  ++i;
}
```

Teste no cling:

```
[cling]$ .L incrementa.c
[cling]$ int i = 0;
[cling]$ incrementa(i)
[cling]$ i
```

# PASSAGEM DE PARÂMETROS

Crie o arquivo incrementa.c

```
void incrementa(int i) {
  ++i;
}
```

Teste no cling:

```
[cling]$ .L incrementa.c
[cling]$ int i = 0;
[cling]$ incrementa(i)
[cling]$ i
(int) 0
[cling]$
```

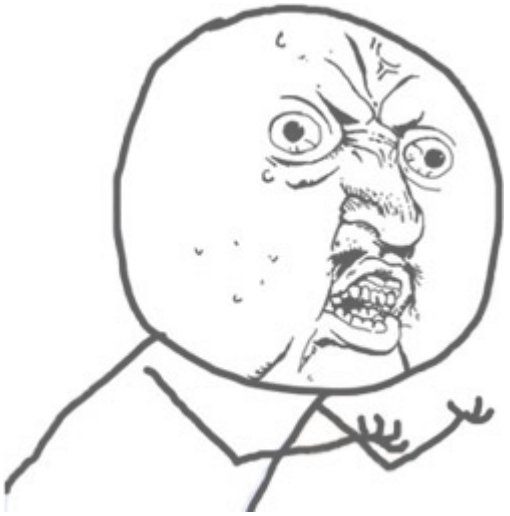
# PASSAGEM DE PARÂMETROS

Crie o arquivo incrementa.c

```
void incrementa(int i) {
  ++i;
}
```

Teste no cling:

```
[cling]$ .L incrementa.c
[cling]$ int i = 0;
[cling]$ incrementa(i)
[cling]$ i
(int) 0
[cling]$
```



*POR QUE VOCÊ FAZ ISSO COMIGO, C !?!?!!!!!11!!UM!!*

# PASSAGEM DE PARÂMETROS

```
void incrementa(int i) {
  ++i;
}
```

- Em C, a passagem de parâmetros para uma função é **sempre por valor**, ou seja, quando passamos uma variável para uma função, o **valor dessa variável é copiado** para o parâmetro correspondente.

- Se quisermos alterar uma variável passada como parâmeto, temos que passar uma **referência** a ela.

# PASSAGEM DE PARÂMETROS

Corrija o arquivo incrementa.c

```
void incrementa(int *i) {
  ++(*i);
}
```

Reinicie o cling e teste:

```
[cling]$ .L incrementa.c
```

# PASSAGEM DE PARÂMETROS

Corrija o arquivo incrementa.c

```
void incrementa(int *i) {
  ++(*i);
}
```

Reinicie o cling e teste:

```
[cling]$ .L incrementa.c
[cling]$ int i = 0;
```

# PASSAGEM DE PARÂMETROS

Corrija o arquivo incrementa.c

```
void incrementa(int *i) {
  ++(*i);
}
```

Reinicie o cling e teste:

```
[cling]$ .L incrementa.c
[cling]$ int i = 0;
[cling]$ incrementa(&i)
```

# PASSAGEM DE PARÂMETROS

Corrija o arquivo incrementa.c

```
void incrementa(int *i) {
  ++(*i);
}
```

Reinicie o cling e teste:

```
[cling]$ .L incrementa.c
[cling]$ int i = 0;
[cling]$ incrementa(&i)
[cling]$
```

# PASSAGEM DE PARÂMETROS

Corrija o arquivo incrementa.c

```
void incrementa(int *i) {
  ++(*i);
}
```

Reinicie o cling e teste:

```
[cling]$ .L incrementa.c
[cling]$ int i = 0;
[cling]$ incrementa(&i)
[cling]$ i
```

# PASSAGEM DE PARÂMETROS

Corrija o arquivo incrementa.c

```
void incrementa(int *i) {
  ++(*i);
}
```

Reinicie o cling e teste:

```
[cling]$ .L incrementa.c
[cling]$ int i = 0;
[cling]$ incrementa(&i)
[cling]$ i
(int) 1
[cling]$
```

# PASSAGEM DE PARÂMETROS

Crie o arquivo ordena.c

Implemente a função de nome troca, que troca os valores de duas variáveis do tipo int.

Exemplo de uso no cling:

```
[cling]$ .L ordena.c
[cling]$ int x = 42, y = 99;
[cling]$ troca(&x, &y);
[cling]$ x
(int) 99
[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]$
```