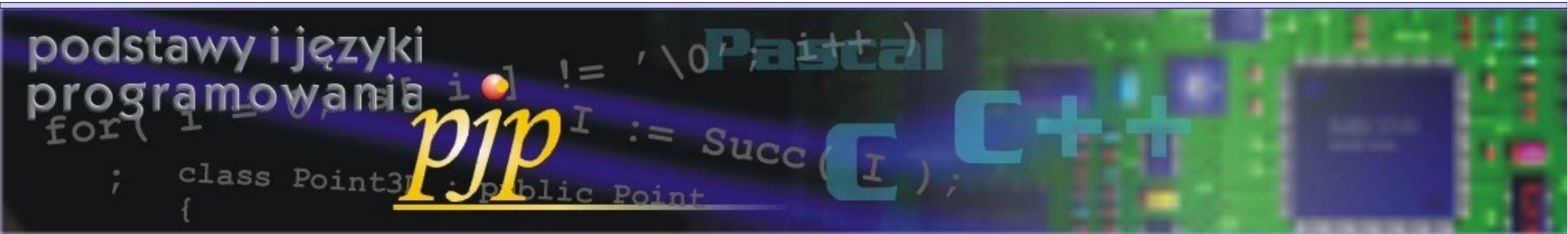
Języki programowania obiektowego
Nieobiektowe elementy języka C++

Roman Simiński
roman.siminski@us.edu.pl
www.programowanie.siminskionline.pl

Przetwarzanie tablic znaków

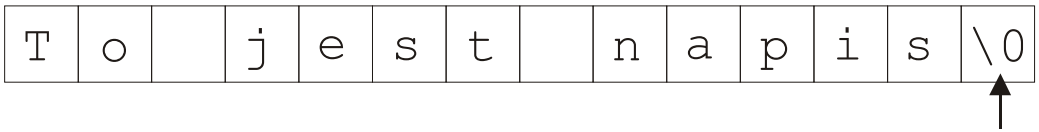


Łańcuchy znakowe jako tablice znaków ze znacznikiem końca

- Do reprezentacji łańcuchów znakowych w języku C/C++ wykorzystuje się zwykłe tablice znakowe.
- Tablice takie nie różnią się od innych tablic w języku C/C++, wprowadzono jedynie *kilka udogodnień*, czyniących łatwiejszym manipulowanie takimi tablicami.

W języku C przyjęto koncepcję łańcuchów ze znacznikiem końca (ang. *null terminated strings*).

`"To jest napis"` a to jego reprezentacja wewnętrzna:



Znacznik końca napisu \0 to znak o kodzie 0

Fizyczna długość napisu = liczba znaków + 1

Deklarowanie i inicjowanie zmiennych łańcuchowych

Tablice znakowe można inicjować w zwykły sposób, przewidziany dla tablic:

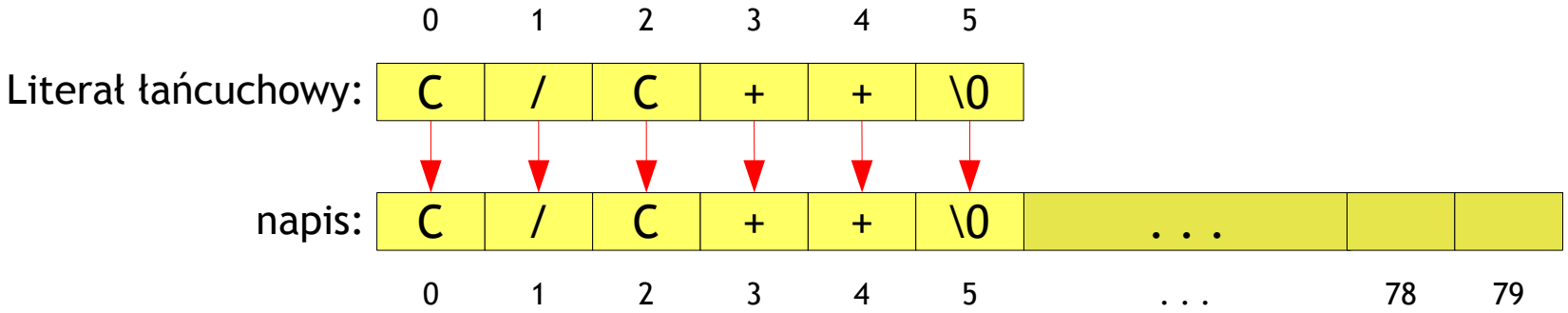
```
const int N = 80;
```

```
char napis[ N ] = { 'C', '/', 'C', '+', '+' };          // \0 ??? Czy dobrze?
```

```
char napis[ N ] = { 'C', '/', 'C', '+', '+', '\0' };    // \0 !!! Lepiej
```

Można wykorzystywać wygodniejszą formę:

```
char napis[ N ] = "C/C++";
```



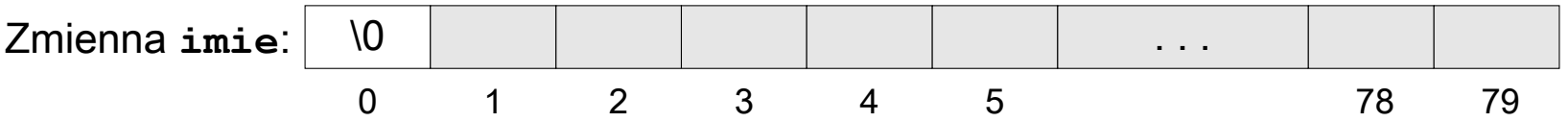
Łańcuch pusty

Deklaracja łańcucha zainicjowanego napisem pustym:

```
char imie[ N ] = { '\0' };
```

```
char imie[ N ] = "";
```

Reprezentacja wewnętrzna łańcucha pustego:



Ustawianie łańcucha pustego po deklaracji:

```
imie[ 0 ] = '\0';
```

Przypisanie literału na etapie inicjalizacji

Uwaga — przypisanie literału łańcuchowego do tablicy znaków może wystąpić tylko przy definiowaniu tablicy i oznacza jej *inicjalizację*.

```
char napis[ N ] = "C/C++";  // OK
. . .
napis = "C/C++";   // Błąd, niedozwolone przypisanie
napis = "";        // Błąd, niedozwolone przypisanie
```

Nie wolno również tak:

```
char napis[ N ] = "C/C++";  // OK
char napis1[ N ] = napis;   // Błąd, niedozwolona inicjalizacja
```

Ogólny schemat przetwarzania tablic znakowych

Przetwarzanie tablic polega zwykle na „przemaszerowaniu” zmienna indeksową po tablicy, dopóki nie ma końca napisu, oznaczanego znakiem '\0':

```
const int N = 80;
int i;
char s[ N ];
. . .
for( i = 0; s[ i ] != '\0'; i++ )
  < tu jakieś operacje na każdym znaku s[ i ] >
```

Wyprowadzanie zawartości napisu *s* do *strumienia wyjściowego* znak po znaku:

```
for( i = 0; s[ i ] != '\0'; i++ )
  cout << s[ i ];
```

Lub krócej:

```
for( i = 0; s[ i ] != '\0'; cout << s[ i++ ] )
  ;
```

Każdy znak napisu w osobnej linii:

```
for( i = 0; s[ i ] != '\0'; cout << endl << s[ i++ ] )
  ;
```

Przetwarzanie z wykorzystaniem funkcji bibliotecznych — string.h

Do manipulowania tablicami znakowymi opracowano szereg funkcji bibliotecznych (plik nagłówkowy *string.h*),... większość z nich można łatwo napisać samemu!

Wyznaczanie długości napisu — funkcja strlen

- Długość napisu to liczba znaków zapisanych w tablicy znakowej, a więc liczba znaków zapisanych przed wystąpieniem znacznika konca napisu.
- Długość napisu to nie rozmiar tablicy a liczba znaków w niej zapisanych.

Rezultatem funkcji *strlen* jest liczba znaków napisu, przekazanego tej funkcji parametrem.

```
const int N = 80;
char napis[ N ] = "Język C++";
. . .
cout << "Liczba znaków w łańcuchu: " << napis << " to: " << strlen( napis );
```

```
Liczba znaków w łańcuchu: Język C++ to: 9
```

Wyznaczanie długości napisu — funkcja strlen, zastosowanie

Alternatywny algorytm przetwarzania tablic znaków:

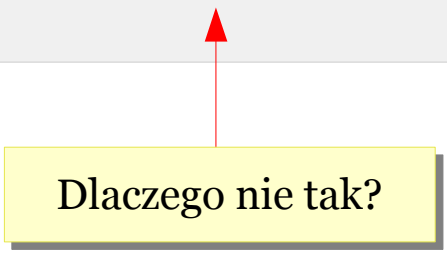
```
int i, dlugosc;
char napis[ N ];
. . .
dlugosc = strlen( napis );
for( i = 0; i < dlugosc; cout << napis[ i++ ] )
  ;
```

Lub:

```
for( i = 0, dlugosc = strlen( napis ); i < dlugosc; cout << napis[ i++ ] )
  ;
```

Ale nie tak:

```
for( i = 0; i < strlen( napis ); cout << napis[ i++ ] )
  ;
```



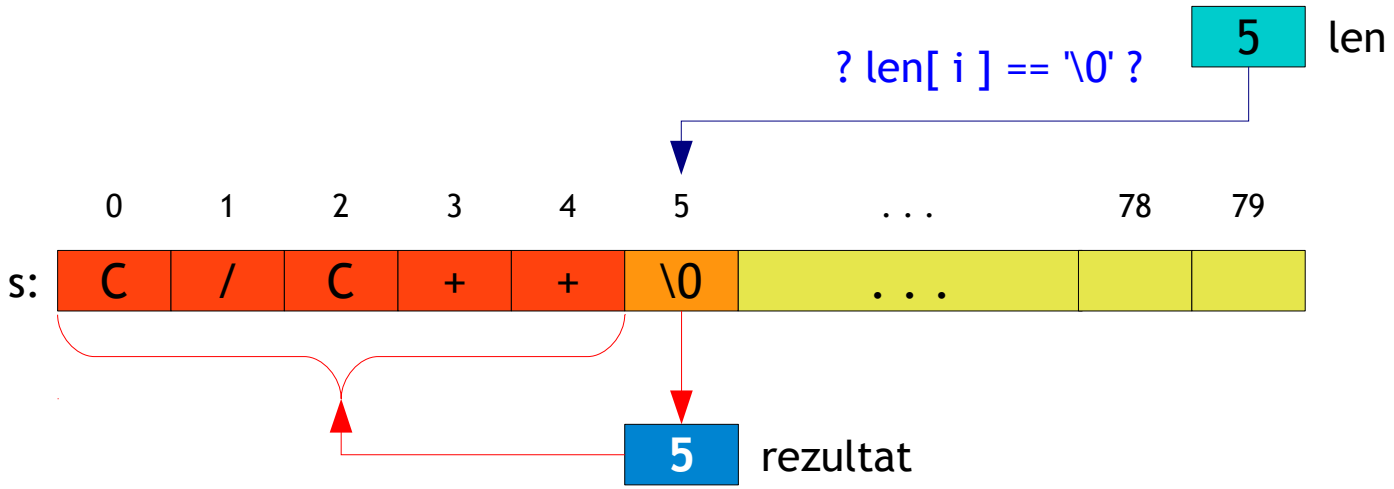
Wyznaczanie długości napisu — funkcja strlen, realizacja

Realizacja — iteracji *while*:

```
int strlen( char s[] )
{
  int len = 0;

  while( s[ len ] != '\0' )
    len++;

  return len;
}
```

Realizacja — iteracji *for*:

```
int strlen( char s[] )
{
  int len;
  for( len = 0; s[ len ] != '\0'; len++ )
    ;
  return len;
}
```



Indeks elementu zawierającego znacznik końca łańcucha określa liczbę znaków go poprzedzających.

Zamiana wielkości liter — funkcje strupr i strlwr

Konwersja - małe litery na duże: *strupr*, duże litery na małe: *strlwr*.

```
char a[ N ] = "ALA";
char b[ N ] = "ala";

strlwr( a ); // Po wywołaniu strlwr zmienna a zawiera napis "ala"
strupr( b ); // Po wywołaniu strupr zmienna a zawiera napis "ALA"
```

Realizacja funkcji *strupr*:

```
void strupr( char s[] )
{
  int i;
  for( i = 0; s[ i ] != '\0'; i++ )
    s[ i ] = toupper( s[ i ] );
}
```

Konwersja elementu tablicy, *toupper* zamienia znak będący parametrem na literę dużą, o ile był literą małą.

Realizacja funkcji *strlwr*:

```
void strlwr( char s[] )
{
  int i;
  for( i = 0; s[ i ] != '\0'; i++ )
    s[ i ] = tolower( s[ i ] );
}
```

Konwersja elementu tablicy, *tolower* zamienia znak będący parametrem na literę małą, o ile był literą dużą.

Kopiowanie napisów — funkcja strcpy, zastosowanie

Pamiętamy, że w językach C/C++ nie można kopiować zawartości tablic wykorzystując operator przypisania.

```
char s1[ 80 ] = "C/C++";
char s2[ 20 ];
. . .
s2 = s1; // Tak nie wolno !!!
```

Do kopiowania zawartości tablic znakowych używamy funkcji *strcpy*

```
strcpy( s2, s1 );
```

- Funkcja *strcpy* kopiuje zawartość tablicy znakowej *s1* do tablicy *s2*. Kopiowaniu podlegają wszystkie znaki łańcucha *s1* (aż do \0), zakłada się, że tablica *s2* ma rozmiar wystarczający na pomieszczenie kopiowanych znaków.
- Funkcja *strcpy* służy również do kopiowania zawartości literałów łańcuchowych:

```
strcpy( s1, "Programowanie " );
strcpy( s2, "w języku C/C++" );
```

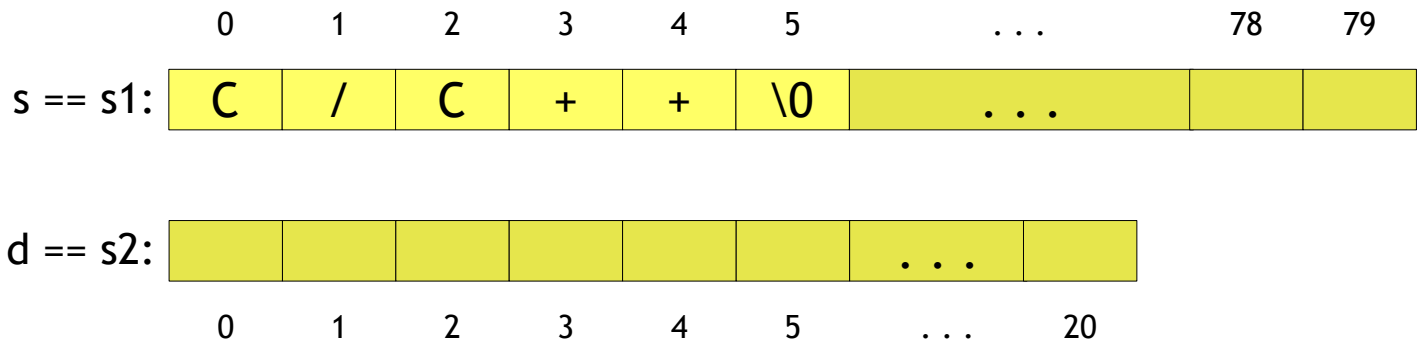
Kopiowanie napisów a parametry tablicowe

```
void strcpy( char d[], char s[] )
{
  . . .;
}
. . .
char s1[ 80 ] = "C/C++";
char s2[ 20 ];
. . .
strcpy( s2, s1 );
```

W C/C++ parametry tablicowe domyślnie przekazywane są przez *zmienną* (*referencję*).

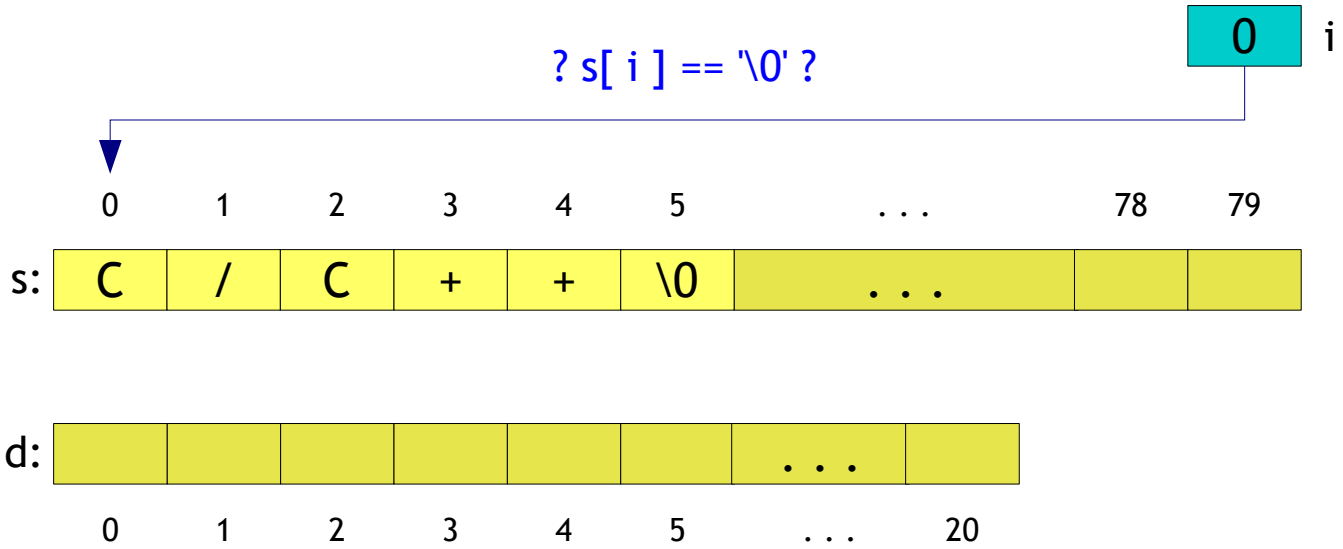
Gdy parametrem jest tablica, we wnętrzu funkcji nie wiadomo ilu elementowa tablica została przekazana jako parametr aktualny wywołania!

Funkcje operujące na napisach poszukują znacznika końca napisu.



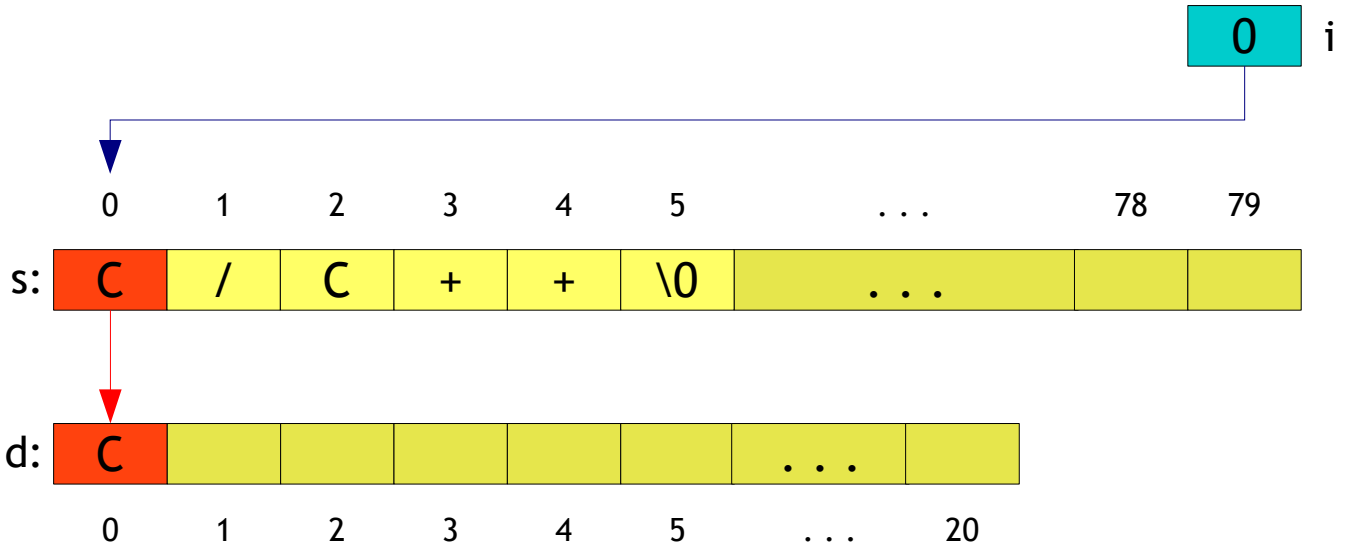
Kopiowanie napisów — funkcja strcpy, przykładowa realizacja

```
void strcpy( char d[], char s[] )
{
  int i = 0;
  while( s[ i ] != '\0' )
  {
    d[ i ] = s[ i ];
    i++;
  }
  d[ i ] = '\0';
}
```



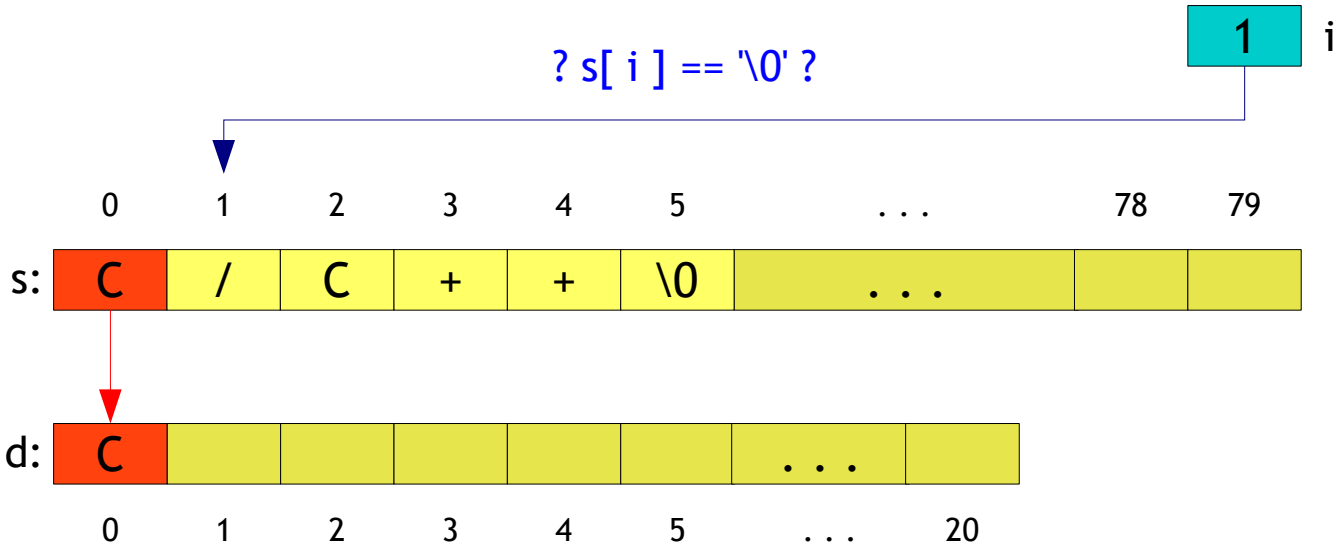
Kopiowanie napisów — funkcja strcpy, przykładowa realizacja

```
void strcpy( char d[], char s[] )
{
  int i = 0;
  while( s[ i ] != '\0' )
  {
    d[ i ] = s[ i ];
    i++;
  }
  d[ i ] = '\0';
}
```



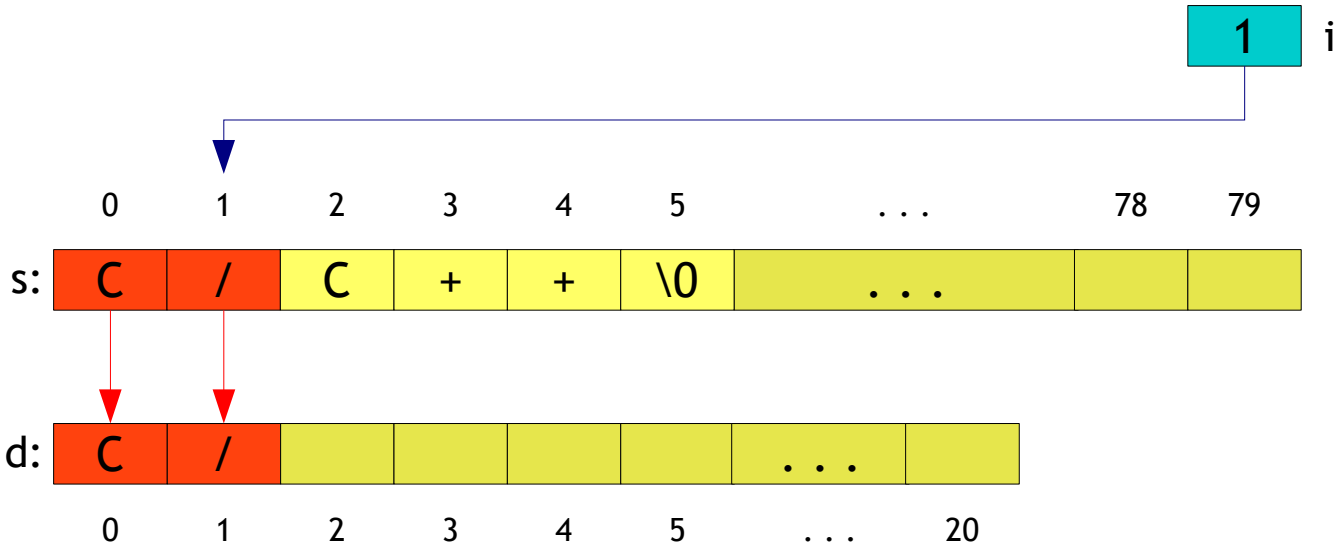
Kopiowanie napisów — funkcja strcpy, przykładowa realizacja

```
void strcpy( char d[], char s[] )
{
  int i = 0;
  while( s[ i ] != '\0' )
  {
    d[ i ] = s[ i ];
    i++;
  }
  d[ i ] = '\0';
}
```



Kopiowanie napisów — funkcja strcpy, przykładowa realizacja

```
void strcpy( char d[], char s[] )
{
  int i = 0;
  while( s[ i ] != '\0' )
  {
    d[ i ] = s[ i ];
    i++;
  }
  d[ i ] = '\0';
}
```

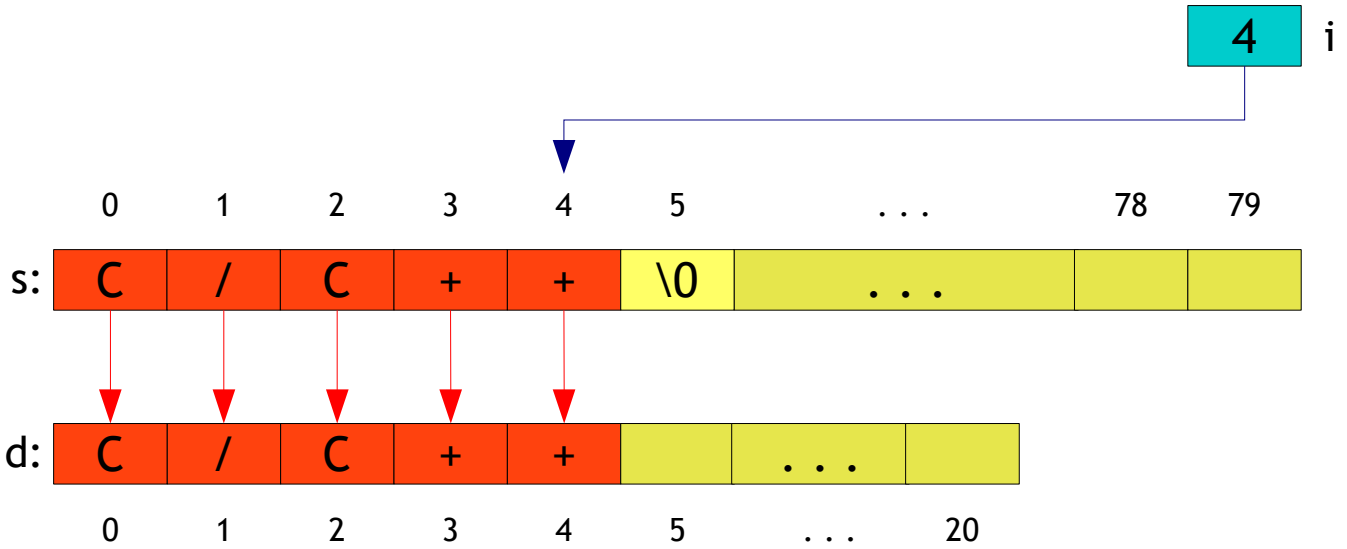


Kopiowanie napisów — funkcja strcpy, przykładowa realizacja

I tak dalej, aż do przepisania ostatniego znaku...

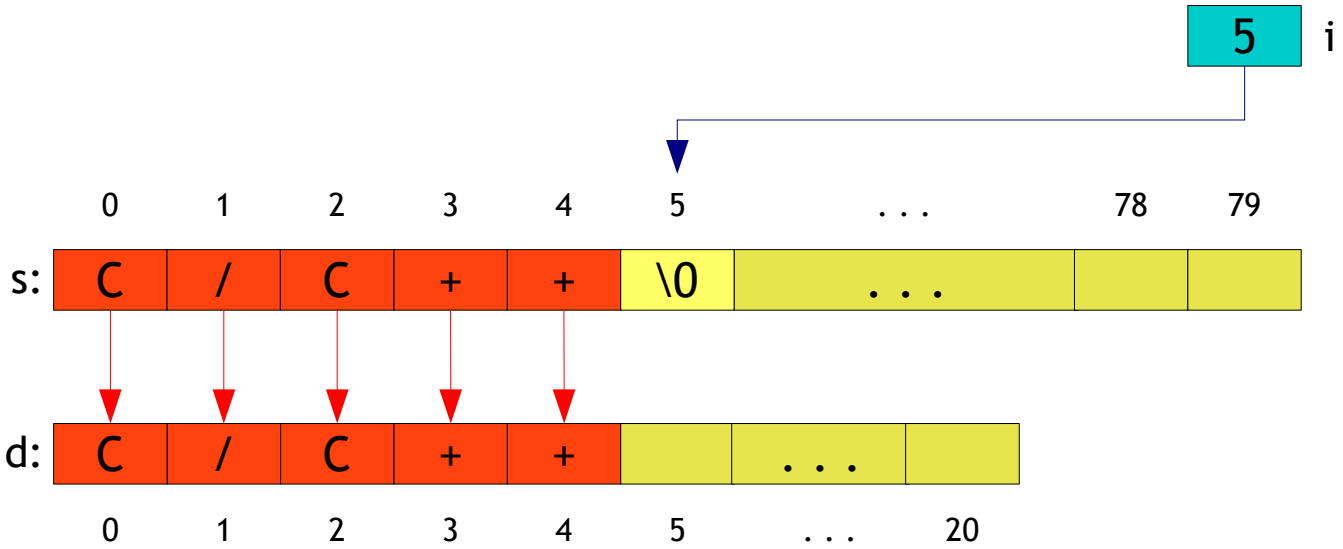
Kopiowanie napisów — funkcja strcpy, przykładowa realizacja

```
void strcpy( char d[], char s[] )
{
  int i = 0;
  while( s[ i ] != '\0' )
  {
    d[ i ] = s[ i ];
    i++;
  }
  d[ i ] = '\0';
}
```



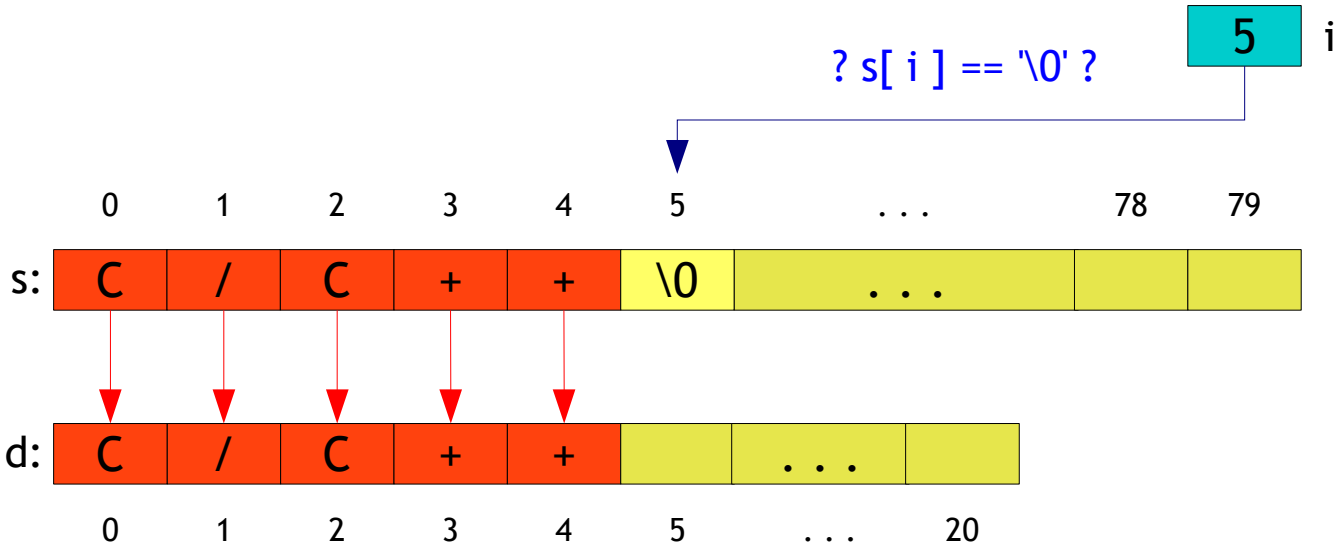
Kopiowanie napisów — funkcja strcpy, przykładowa realizacja

```
void strcpy( char d[], char s[] )
{
  int i = 0;
  while( s[ i ] != '\0' )
  {
    d[ i ] = s[ i ];
    i++;
  }
  d[ i ] = '\0';
}
```



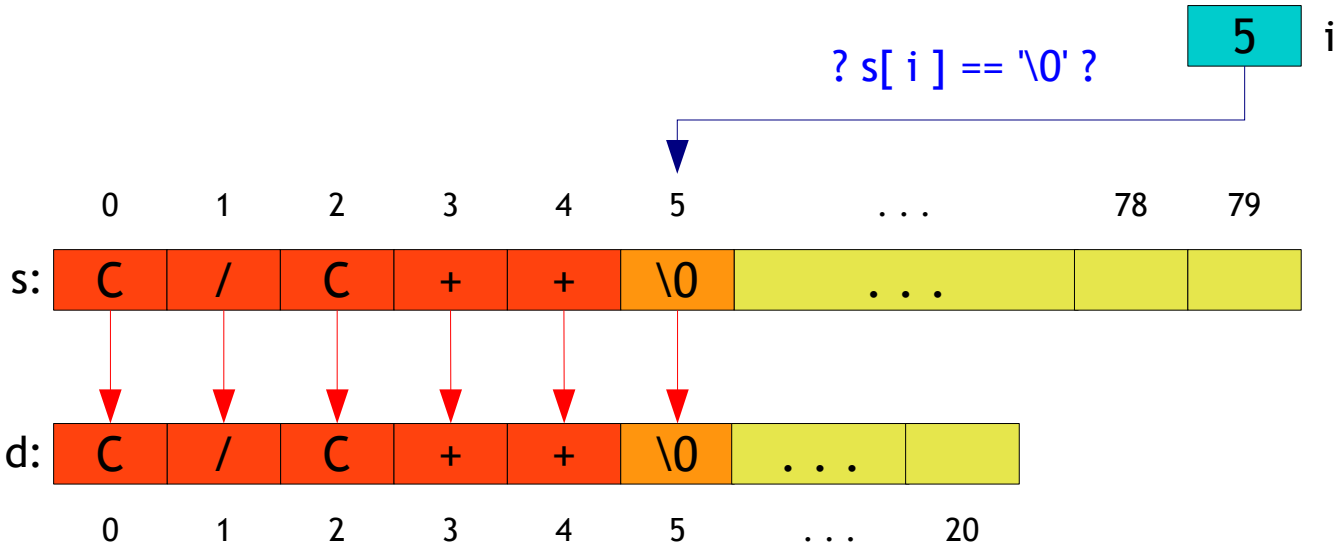
Kopiowanie napisów — funkcja strcpy, przykładowa realizacja

```
void strcpy( char d[], char s[] )
{
  int i = 0;
  while( s[ i ] != '\0' )
  {
    d[ i ] = s[ i ];
    i++;
  }
  d[ i ] = '\0';
}
```



Kopiowanie napisów — funkcja strcpy, przykładowa realizacja

```
void strcpy( char d[], char s[] ))
{
  int i = 0;
  while( s[ i ] != '\0' )
  {
    d[ i ] = s[ i ];
    i++;
  }
  d[ i ] = '\0';
}
```



Kopiowanie napisów — funkcja strcpy, wersja uproszczona

W językach C/C++ operator przypisania jest *lewostronnie łączny*. Pozwala to na pisanie następujących konstrukcji:

```
d[ 0 ] = d[ 1 ] = d[ 2 ] = 0;
```

Alternatywna, uproszczona realizacja kopiowania napisów:

```
void strcpy( char d[], char s[] )
{
  int i = 0;
  while( ( d[ i ] = s[ i ] ) != '\0' )
    i++;
}
```

Konstrukcja:

```
( d[ i ] = s[ i ] ) != '\0'
```

Przypisuje *i*-ty element tablicy *s* do *i*-tego elementu tablicy *d*. Przypisana wartość jest następnie porównywana (operator !=) ze znacznikiem końca napisu '\0'.

Ta wersja funkcji *strcpy* przepisuje znacznik końca napisu z tablicy *s* do *d* w iteracji while.

Łączenie napisów — funkcja strcat

Funkcja *strcat* dołącza zawartość tablicy znakowej *s1* do tablicy *s2*. Kopiowaniu podlegają wszystkie znaki łańcucha *s1* (aż do \0), zakłada się, że tablica *s2* ma rozmiar wystarczający na pomieszczenie dołączanych znaków.

```
strcpy( s1, "Programowanie " );
strcpy( s2, "w języku C/C++" );

strcat( s1, s2 );
cout << s1;
```

→ `Programowanie w języku C/C++`

Jak to działa?

```
void strcat( char d[], char s[] )
{
  int i = 0, j = 0;

  while( d[ i ] != '\0' )
    i++;

  while( ( d[ i++ ] = s[ j++ ] ) != '\0' )
      ;
}
```

Znajdź znacznik końca napisu docelowego, zapamiętaj jego pozycje w zmiennej *i*.

Przepisz elementy tablicy *s* do tablicy *d*. Maszeruj zmienną *j* od początku tablicy *s*, zmienną *i* od pozycji znalezionego wcześniej znacznika końca napisu.

Co się stanie, gdy tablica docelowa jest za krótka?

```
int main()
{
  char s1[ 5 ] = "AAAA";
  char c1 = 'A';
  char c2 = 'B';
  char s2[ 5 ] = "BBBB";

  strcpy( s2, "XXXXXXXXXXXXXXXXXXX" );

  cout << "\ns1 :" << s1;
  cout << "\nc1 :" << c1;
  cout << "\ns2 :" << s2;
  cout << "\nc2 :" << c2;
  . . .
}
```

```
s1 :XXXXXXXXXXXXXXX
c1 :X
s2 :XXXXXXXXXXXXXXXXXXX
c2 :X
```

- Gdzie oryginalna zawartość tablicy **s1**?
- Co się stało ze zmienną **c1**, **c2**?

Nigdy nie należy zakładać, że „się uda”, czyli że tablica docelowa jest *wystarczająco długa*. Należy szacować, przewidywać, jeszcze raz przewidywać — *programować defensywnie*.

Jak nie dopuszczać do „przepełnienia bufora”?

Biblioteka funkcji operujących na tablicach znaków zawiera funkcje wykonujące operacje analogiczne do przedstawionych uprzednio, pozwalające na kontrolę liczby znaków biorących udział np. w kopiowaniu. Są to np. funkcje: *strncpy*, *strncat*, itp.

```
const int N = 10;
const int M = 80;
char s1[ N ];
char s2[ M ] = "Język C jest świetny lecz pełen pułapek";
```

```
strncpy( s1, s2, N - 1 );
s1[ N - 1 ] = '\0';
cout << s1;
```

```
strncpy( s1, s2, sizeof( s1 ) - 1 );
s1[ sizeof( s1 ) - 1 ] = '\0';
cout << s1;
```

Funkcja **`strncpy`** nie zawsze przekopiuje **`'\0'`**!

Na marginesie...

Często rezultatem funkcji operujących na tablicach są one same. Np. rezultatem funkcji **`strcpy`**, **`strcat`**, **`strncpy`** jest tablica będąca pierwszym parametrem wywołania.

```
strncpy( s1, s2, N - 1 );
s1[ N - 1 ] = '\0'
```

```
strncpy( s1, s2, N - 1 )[ N - 1 ] = '\0';
```

s1

Skrócona wersja kopiowania i dopisywania znacznika końca napisu

```
strcpy( s1, "Jezyk C" );
strcat( s1, "i C++" );
strcat( s1, "dla profi!" );
strcpy( s2, s1 );

strcpy( s2, strcat( strcat( strcpy( s1, "Jezyk C" ), "i C++" ), "dla profi!" ) );
```

```
strcpy( s2, strcat( strcat( strcpy( s1, "Jezyk C" ), "i C++" ), "dla profi!" ) );
```

Funkcja strncpy — przykładowa, bezpieczniejsza realizacja

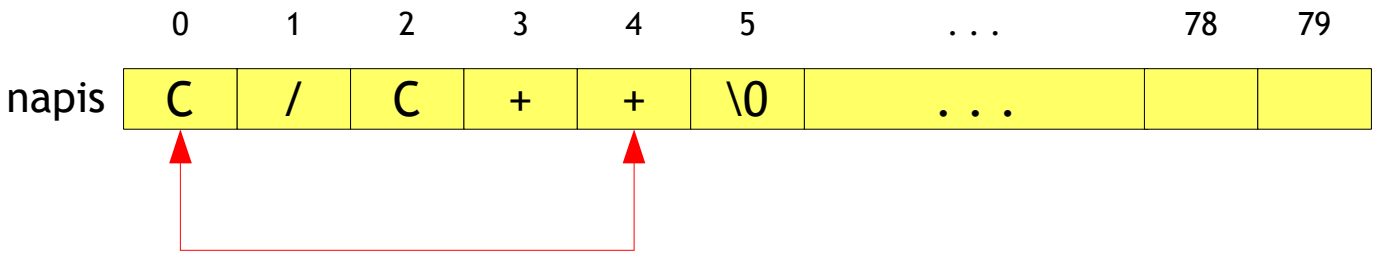
```
void strncpy_while( char d[], char s[], int n )
{
  int i = 0;
  while( ( d[ i ] = s[ i ] ) != '\0' && i < n )
    i++;
  while( i <= n )
    d[ i++ ] = '\0';
}
```

```
void strncpy_for( char d[], char s[], int n )
{
  int i = 0;
  for( ; ( d[ i ] = s[ i ] ) != '\0' && i < n ; i++ )
    ;
  for( ; i <= n ; d[ i++ ] = '\0' )
    ;
}
```

Odwracanie kolejności znaków w napisie — strrev

```
const int MAKS_DL = 80;
char napis[ MAKS_DL ] = "C/C++";

cout << endl << napis;
strrev( napis );
cout << endl << napis;
```

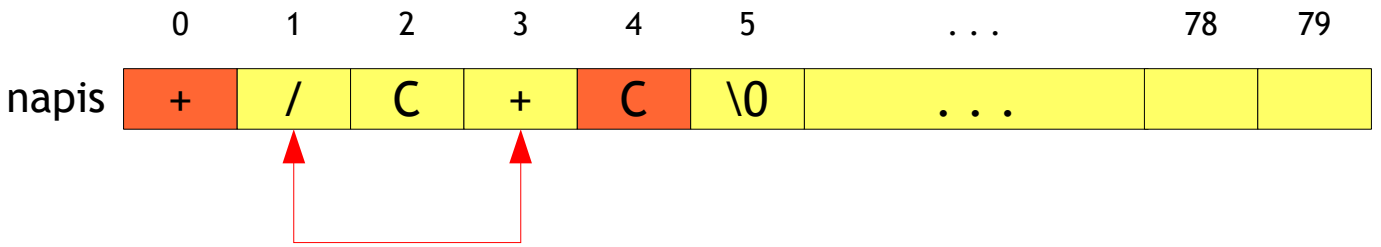
```
C/C++
++C/C
```



Odwracanie kolejności znaków w napisie — strrev

```
const int MAKS_DL = 80;
char napis[ MAKS_DL ] = "C/C++";

cout << endl << napis;
strrev( napis );
cout << endl << napis;
```

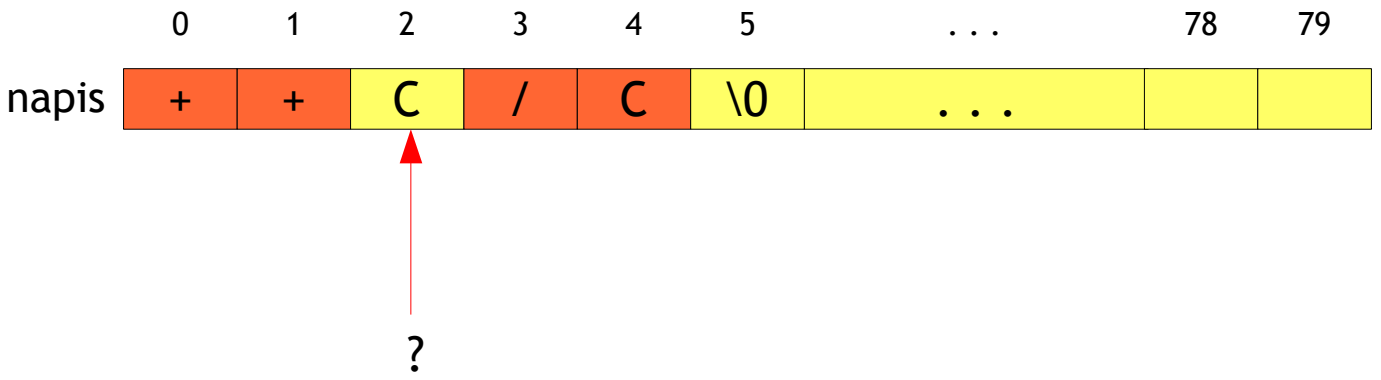
```
C/C++
++C/C
```



Odwracanie kolejności znaków w napisie — strrev

```
const int MAKS_DL = 80;
char napis[ MAKS_DL ] = "C/C++";

cout << endl << napis;
strrev( napis );
cout << endl << napis;
```

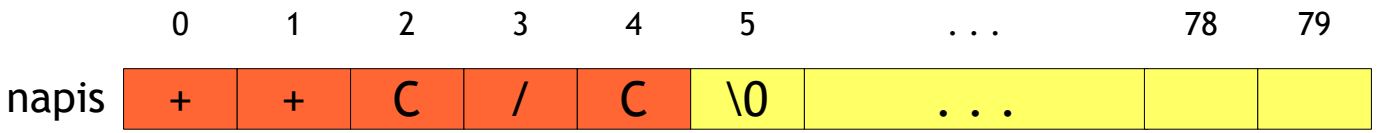
```
C/C++
++C/C
```



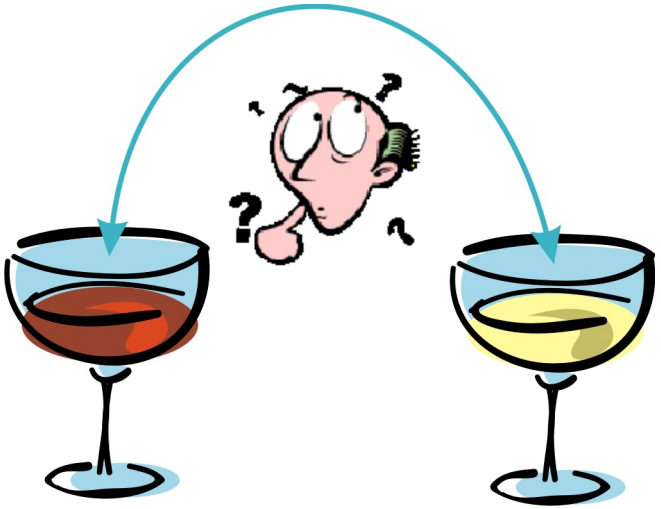
Odwracanie kolejności znaków w napisie — strrev

```
const int MAKS_DL = 80;
char napis[ MAKS_DL ] = "C/C++";

cout << endl << napis;
strrev( napis );
cout << endl << napis;
```

```
C/C++
++C/C
```

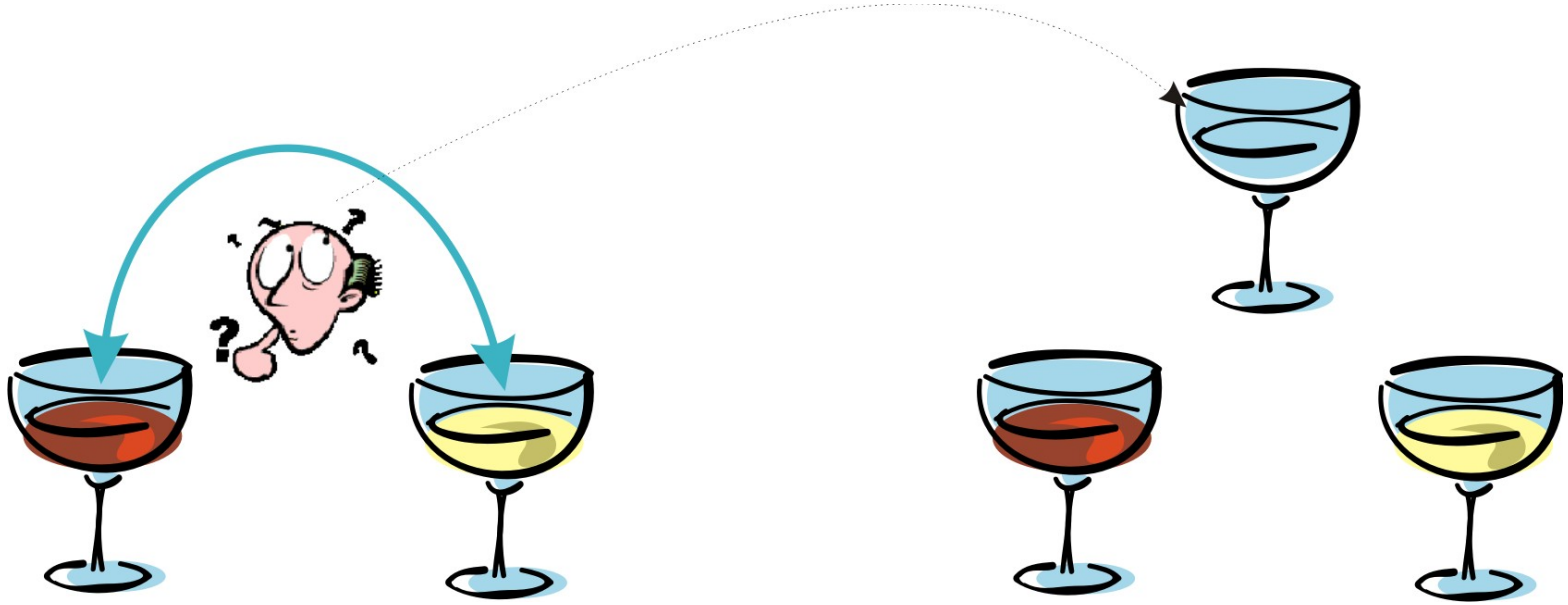


Na marginesie — zamiana wartości w zmiennych, jak?



```
int a = 5;
int b = 10;

a = b;
b = a;
```

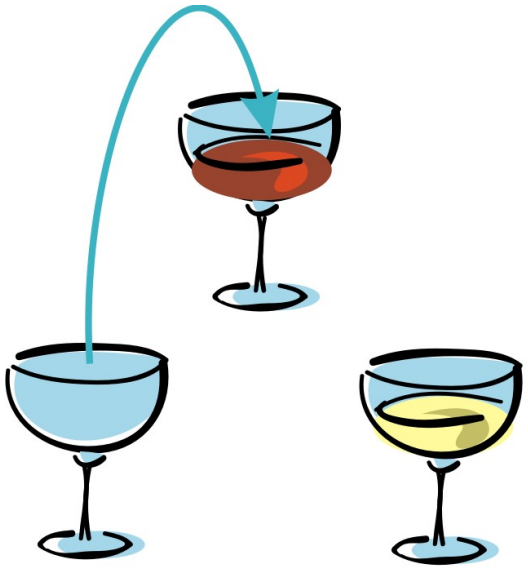
Na marginesie — zamiana wartości w zmiennych, jak?



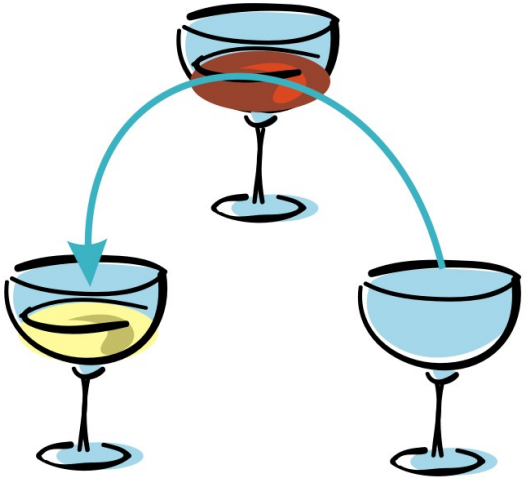
```
int a = 5;
int b = 10;

a = b;
b = a;
```

```
int a = 5;
int b = 10;
int c;
```

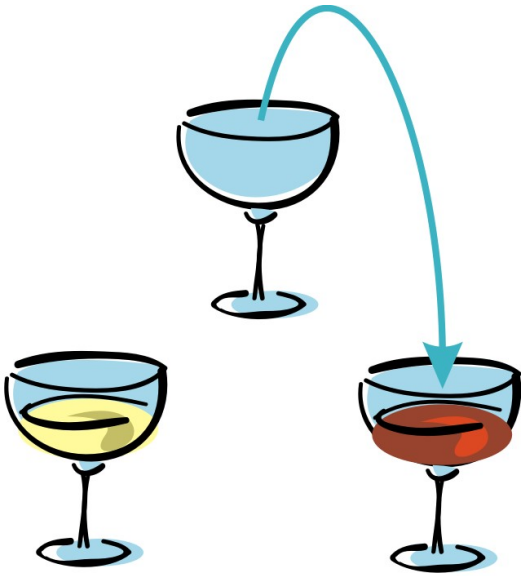
Na marginesie — zamiana wartości w zmiennych, tak...



1



2



3

```
int a = 5;
int b = 10;
int c;

c = a; // 1
a = b; // 2
b = c; // 3
```

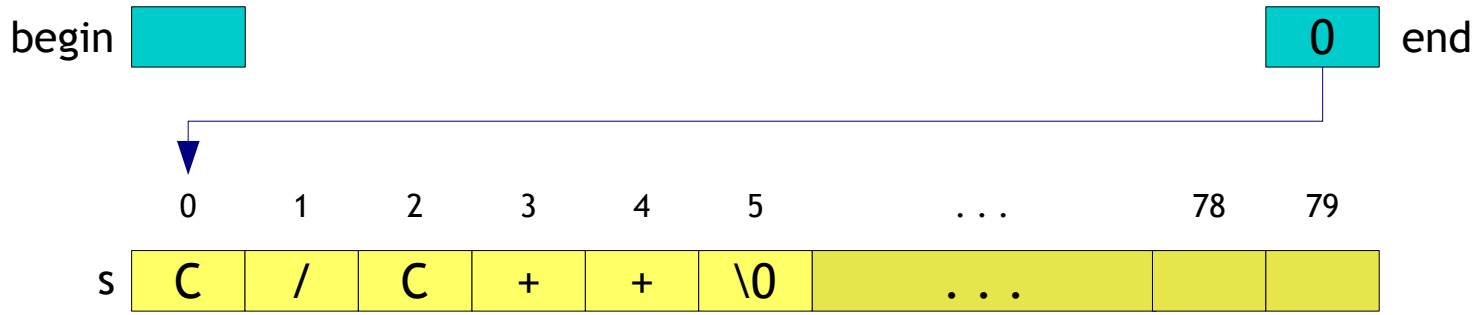
Ciekawostka — zamiana dla zmiennych numerycznych

```
int a = 5;
int b = 10;
int c;

b = b + a;
a = b – a;
b = b – a;
```

To pozornie sprytne rozwiązanie ma subtelną wadę — dla liczb bliskich wartości granicznych zakresu typu może dojść do obcięć, a dla zmiennych rzeczywistych do zaokrągleń. W efekcie wartości po zamianie mogą nie być identyczne!

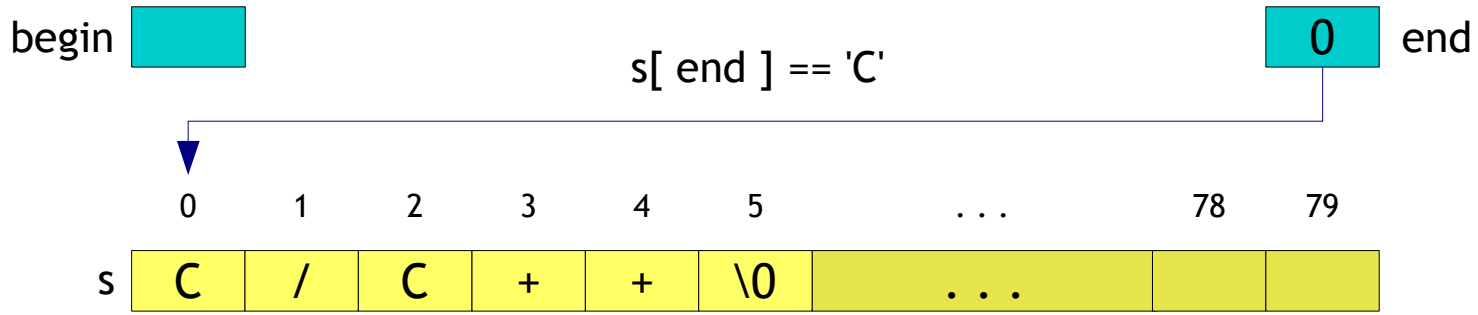
Odwracanie kolejności znaków w napisie — strrev

```
void strrev( char s[] )
{
  int begin, end;

  // Szukanie konca napisu
  for( end = 0; s[ end ] != '\0'; end++ )
    ;

  // . . .
}
```



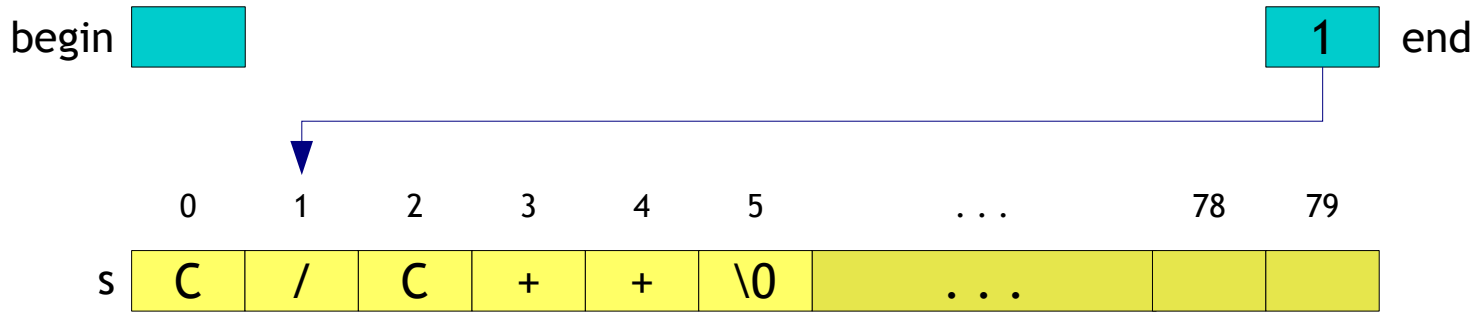
Odwracanie kolejności znaków w napisie — strrev

```
void strrev( char s[] )
{
  int begin, end;

  // Szukanie konca napisu
  for( end = 0; s[ end ] != '\0'; end++ )
    ;

  // . . .
}
```



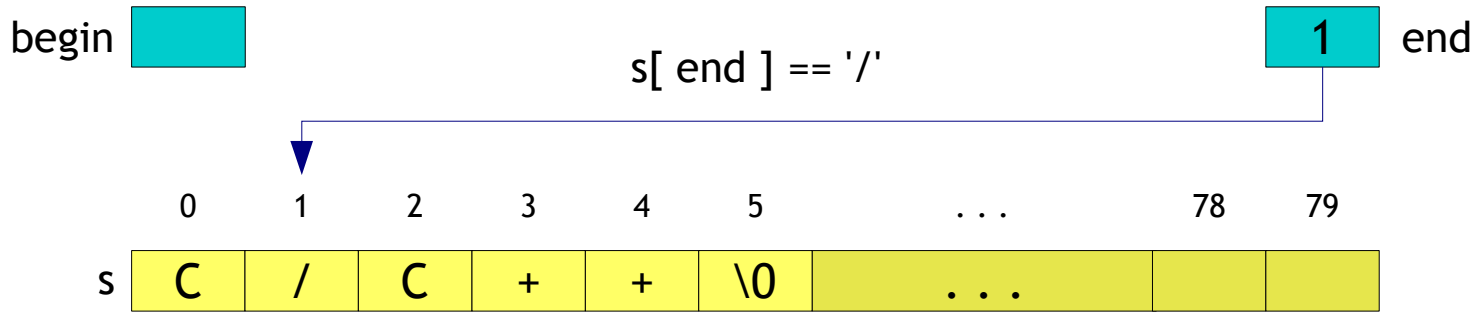
Odwracanie kolejności znaków w napisie — strrev

```
void strrev( char s[] )
{
  int begin, end;

  // Szukanie konca napisu
  for( end = 0; s[ end ] != '\0'; end++ )
    ;

  // . . .
}
```



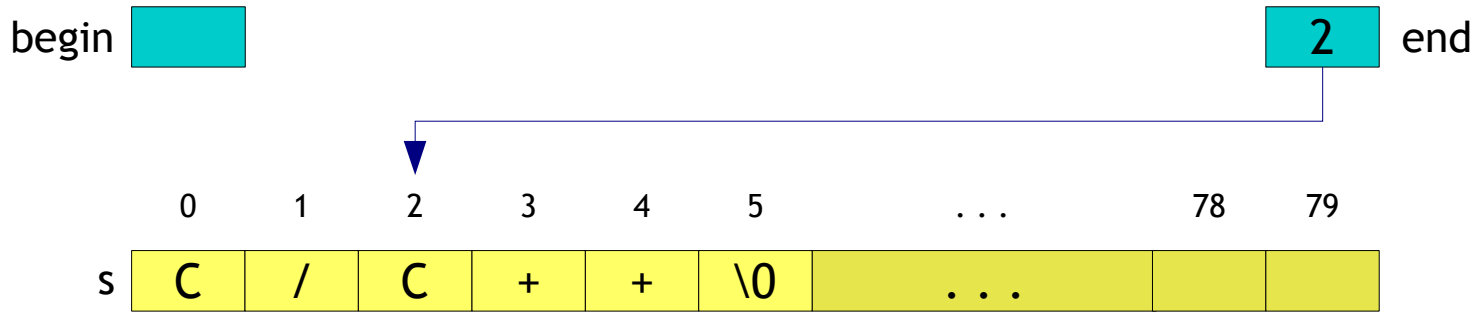
Odwracanie kolejności znaków w napisie — strrev

```
void strrev( char s[] )
{
  int begin, end;

  // Szukanie konca napisu
  for( end = 0; s[ end ] != '\0'; end++ )
    ;

  // . . .
}
```



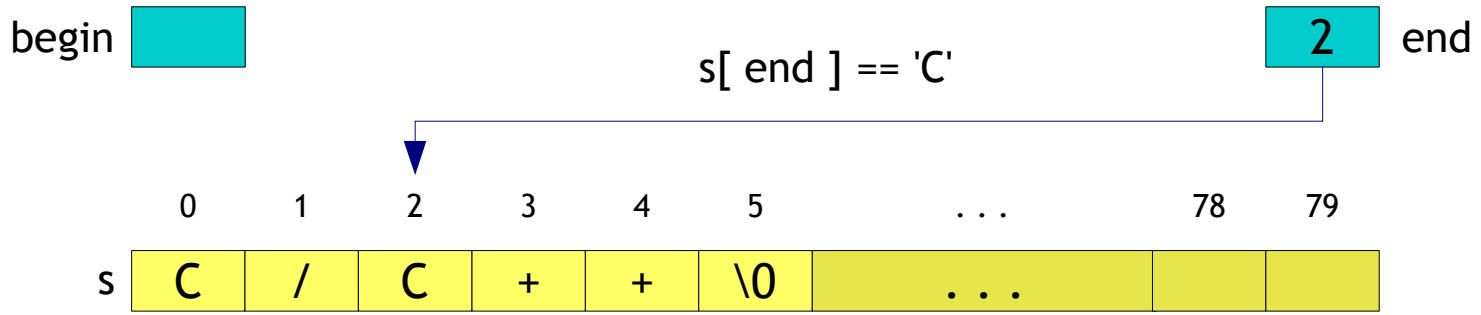
Odwracanie kolejności znaków w napisie — strrev

```
void strrev( char s[] )
{
  int begin, end;

  // Szukanie konca napisu
  for( end = 0; s[ end ] != '\0'; end++ )
    ;

  // . . .
}
```



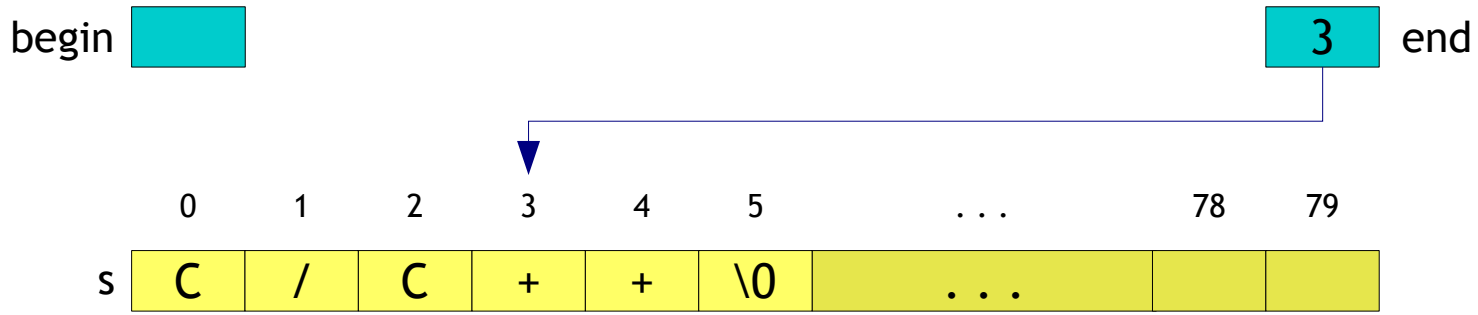
Odwracanie kolejności znaków w napisie — strrev

```
void strrev( char s[] )
{
  int begin, end;

  // Szukanie konca napisu
  for( end = 0; s[ end ] != '\0'; end++ )
    ;

  // . . .
}
```



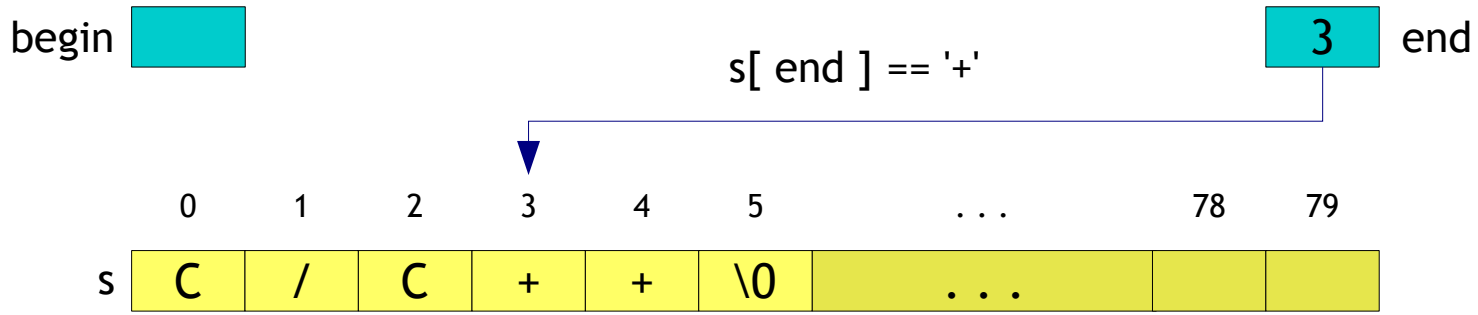
Odwracanie kolejności znaków w napisie — strrev

```
void strrev( char s[] )
{
  int begin, end;

  // Szukanie konca napisu
  for( end = 0; s[ end ] != '\0'; end++ )
    ;

  // . . .
}
```



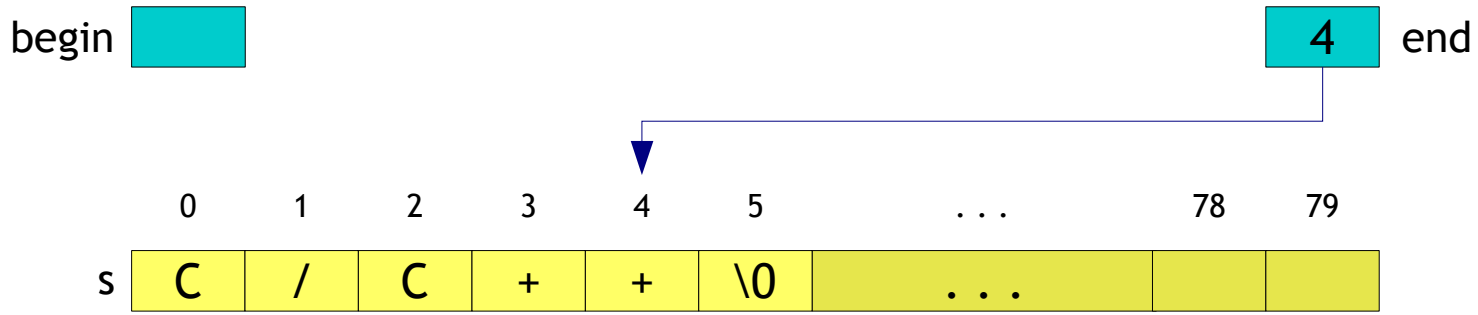
Odwracanie kolejności znaków w napisie — strrev

```
void strrev( char s[] )
{
  int begin, end;

  // Szukanie konca napisu
  for( end = 0; s[ end ] != '\0'; end++ )
    ;

  // . . .
}
```



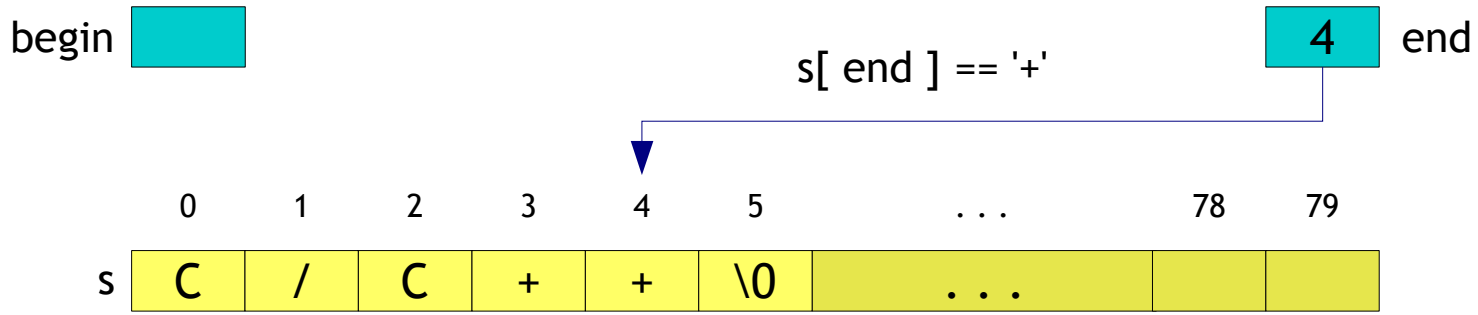
Odwracanie kolejności znaków w napisie — strrev

```
void strrev( char s[] )
{
  int begin, end;

  // Szukanie konca napisu
  for( end = 0; s[ end ] != '\0'; end++ )
    ;

  // . . .
}
```



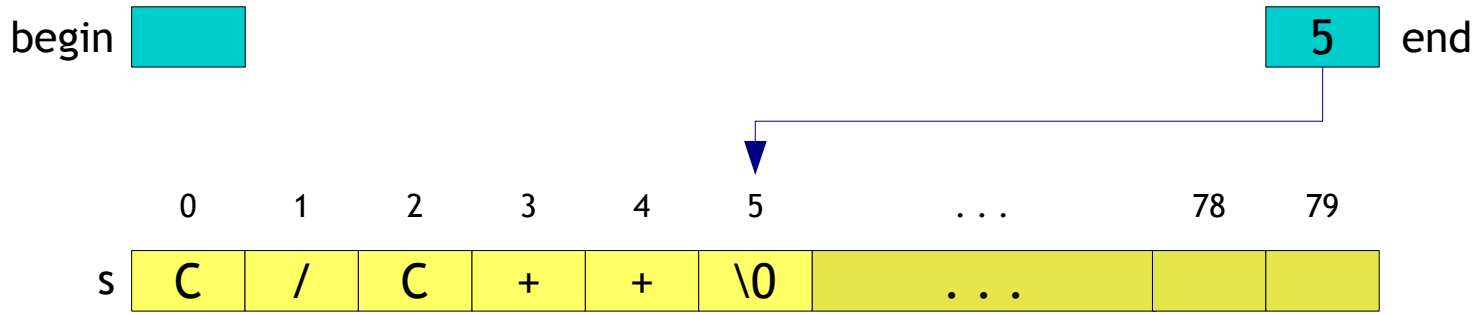
Odwracanie kolejności znaków w napisie — strrev

```
void strrev( char s[] )
{
  int begin, end;

  // Szukanie konca napisu
  for( end = 0; s[ end ] != '\0'; end++ )
    ;

  // . . .
}
```



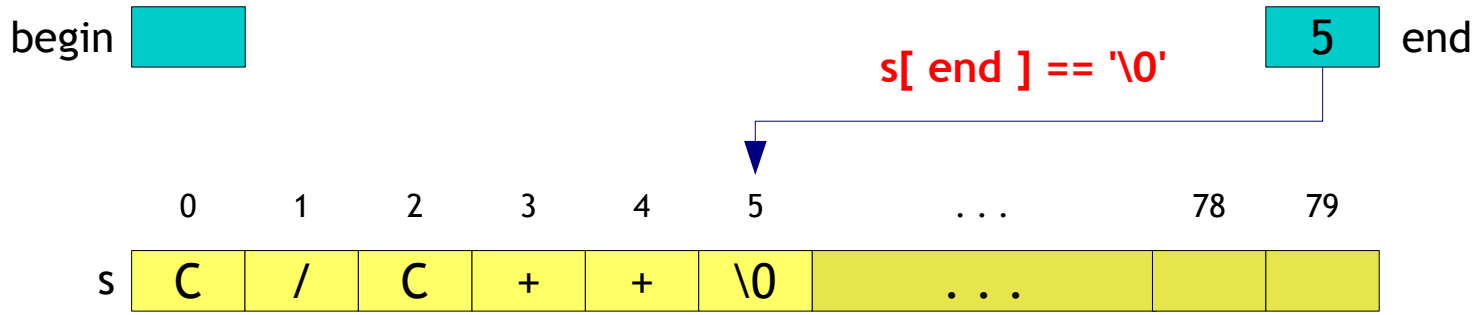
Odwracanie kolejności znaków w napisie — strrev

```
void strrev( char s[] )
{
  int begin, end;

  // Szukanie konca napisu
  for( end = 0; s[ end ] != '\0'; end++ )
    ;

  // . . .
}
```



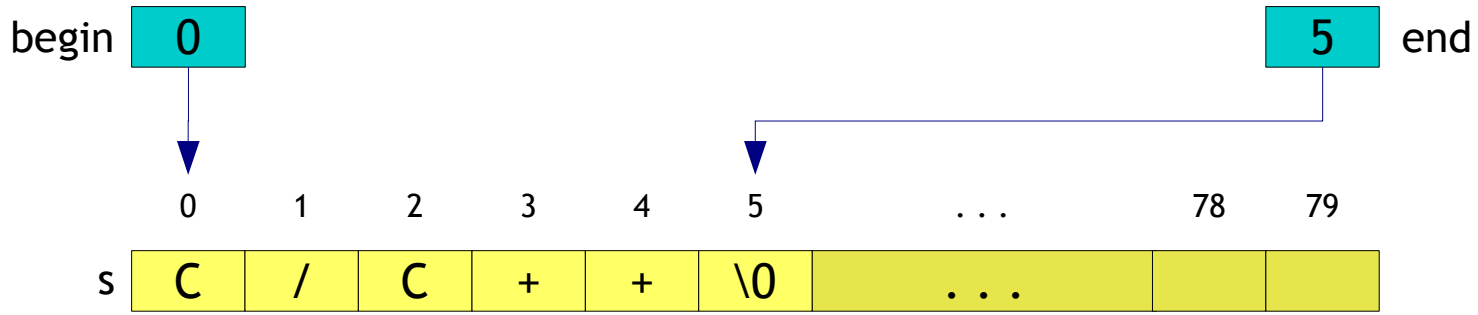
Odwracanie kolejności znaków w napisie — strrev

```
void strrev( char s[] )
{
  int begin, end;

  // Szukanie konca napisu
  for( end = 0; s[ end ] != '\0'; end++ )
    ;

  // . . .
}
```



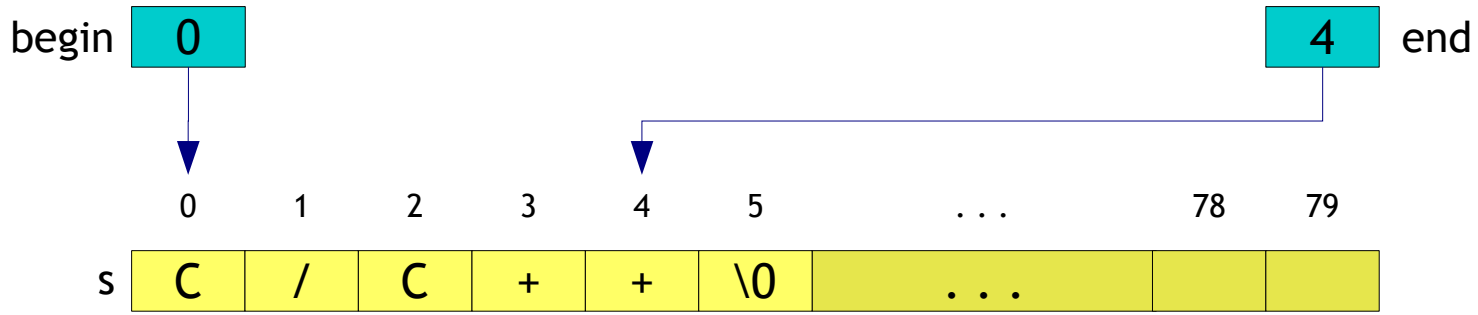
Odwracanie kolejności znaków w napisie — strrev

```
void strrev( char s[] )
{
  int begin, end;
  . . .
  // Zamiana znakow miejscami
  for( begin = 0, end--; begin < end; begin++, end-- )
  {
    char c = s[ begin ];
    s[ begin ] = s[ end ];
    s[ end ] = c;
  }
}
```



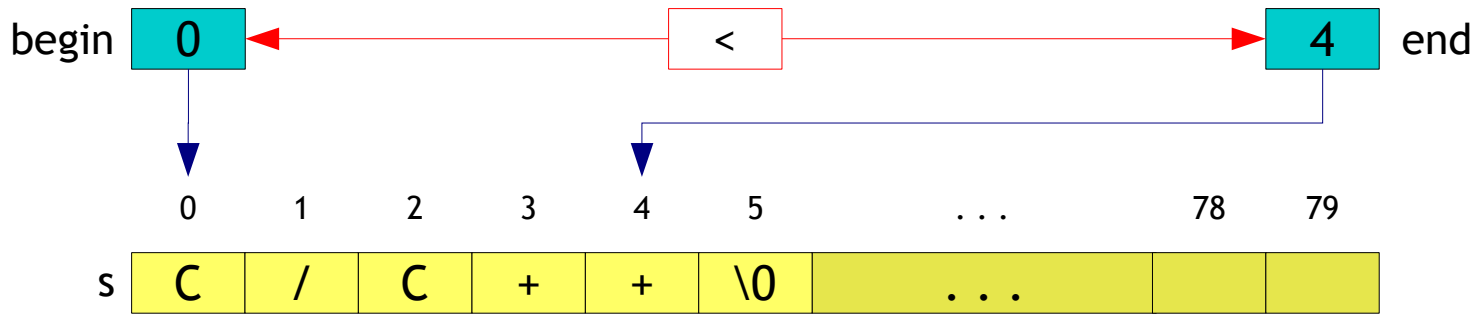
Odwracanie kolejności znaków w napisie — strrev

```
void strrev( char s[] )
{
  int begin, end;
  . . .
  // Zamiana znakow miejscami
  for( begin = 0, end--; begin < end; begin++, end-- )
  {
    char c = s[ begin ];
    s[ begin ] = s[ end ];
    s[ end ] = c;
  }
}
```



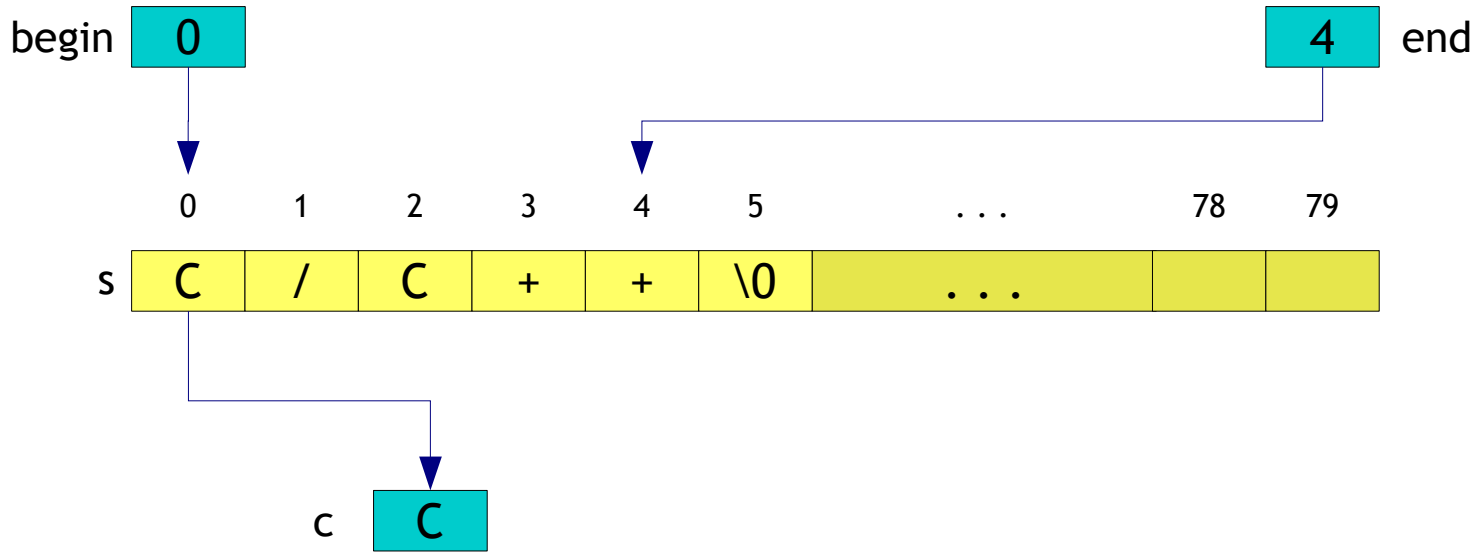
Odwracanie kolejności znaków w napisie — strrev

```
void strrev( char s[] )
{
  int begin, end;
  . . .
  // Zamiana znakow miejscami
  for( begin = 0, end--; begin < end; begin++, end-- )
  {
    char c = s[ begin ];
    s[ begin ] = s[ end ];
    s[ end ] = c;
  }
}
```



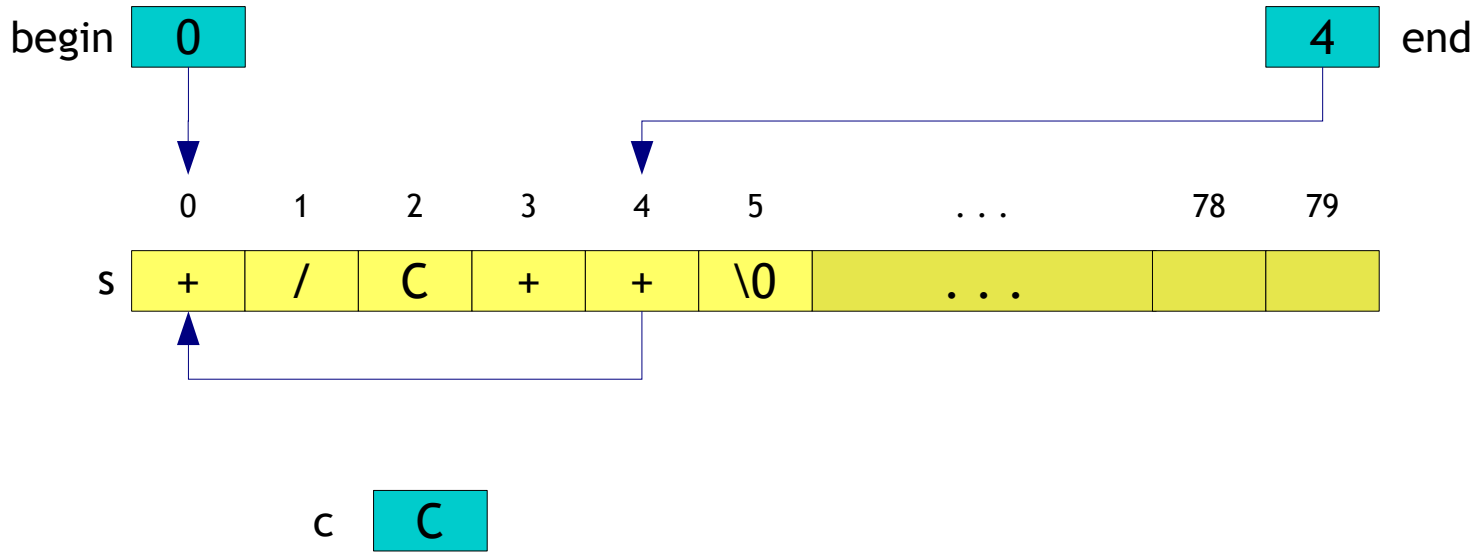
Odwracanie kolejności znaków w napisie — strrev

```
void strrev( char s[] )
{
  int begin, end;
  . . .
  // Zamiana znakow miejscami
  for( begin = 0, end--; begin < end; begin++, end-- )
  {
    char c = s[ begin ];
    s[ begin ] = s[ end ];
    s[ end ] = c;
  }
}
```



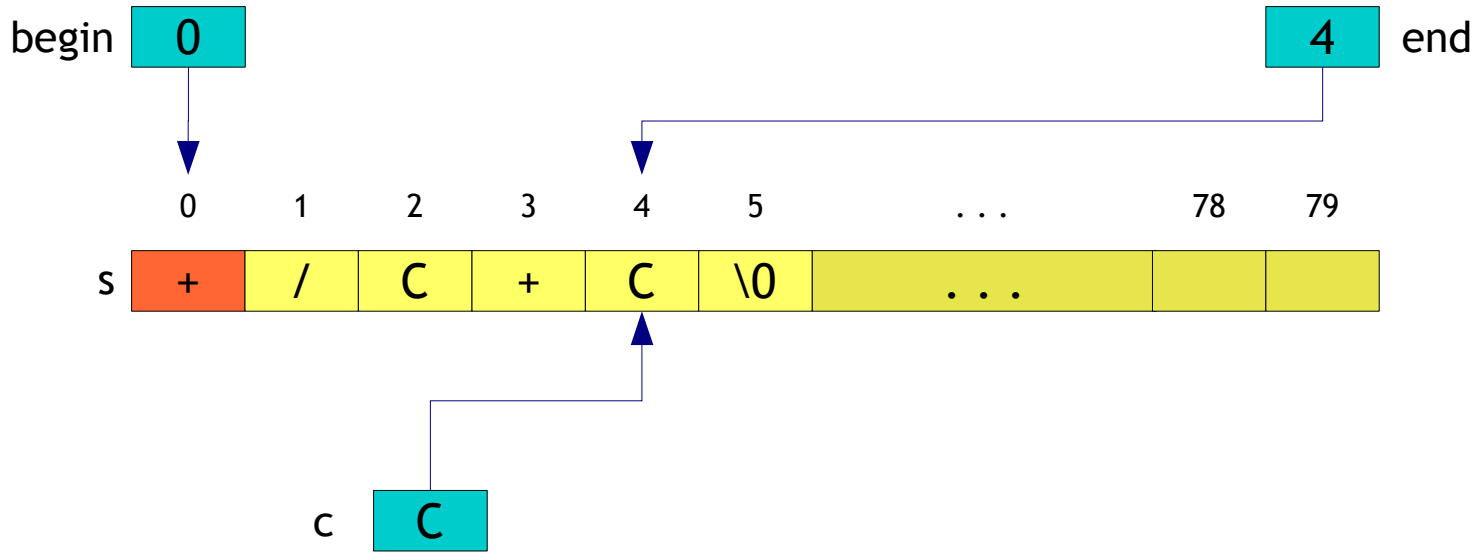
Odwracanie kolejności znaków w napisie — strrev

```
void strrev( char s[] )
{
  int begin, end;
  . . .
  // Zamiana znakow miejscami
  for( begin = 0, end--; begin < end; begin++, end-- )
  {
    char c = s[ begin ];
    s[ begin ] = s[ end ];
    s[ end ] = c;
  }
}
```



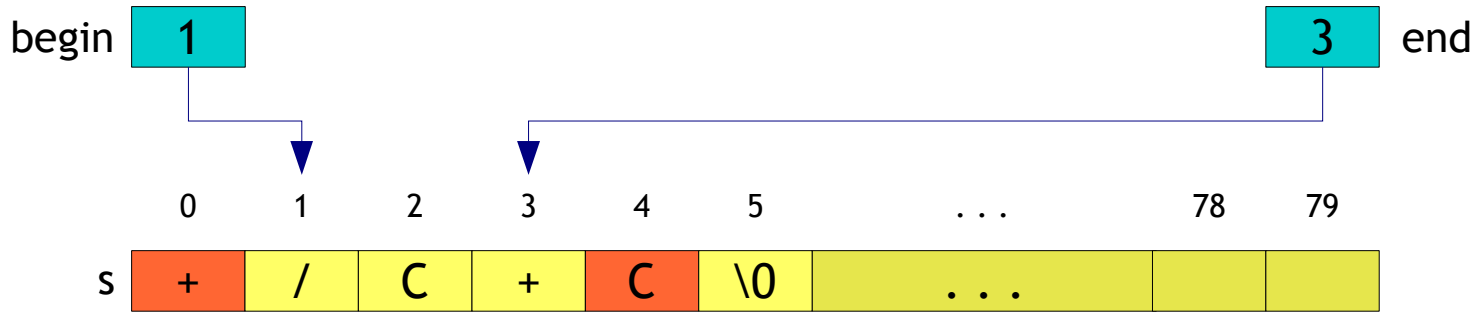
Odwracanie kolejności znaków w napisie — strrev

```
void strrev( char s[] )
{
  int begin, end;
  . . .
  // Zamiana znakow miejscami
  for( begin = 0, end--; begin < end; begin++, end-- )
  {
    char c = s[ begin ];
    s[ begin ] = s[ end ];
    s[ end ] = c;
  }
}
```



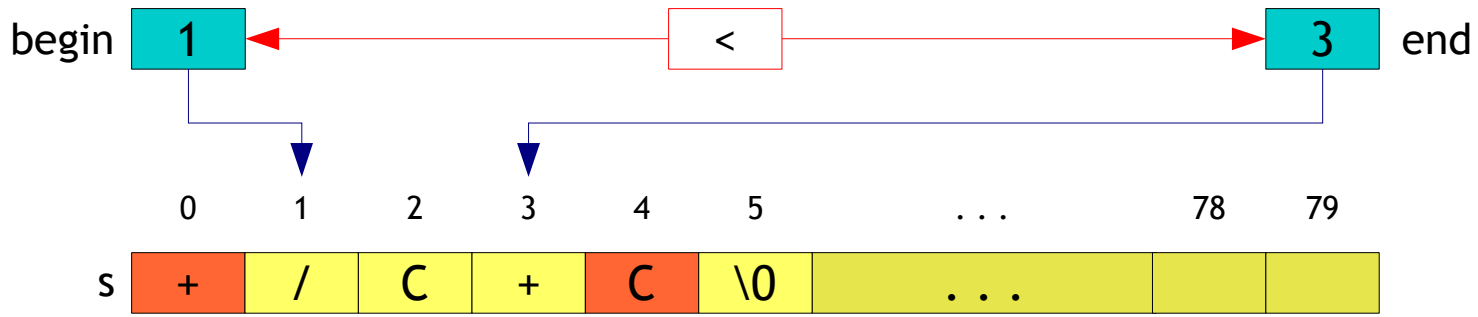
Odwracanie kolejności znaków w napisie — strrev

```
void strrev( char s[] )
{
  int begin, end;
  . . .
  // Zamiana znakow miejscami
  for( begin = 0, end--; begin < end; begin++, end-- )
  {
    char c = s[ begin ];
    s[ begin ] = s[ end ];
    s[ end ] = c;
  }
}
```



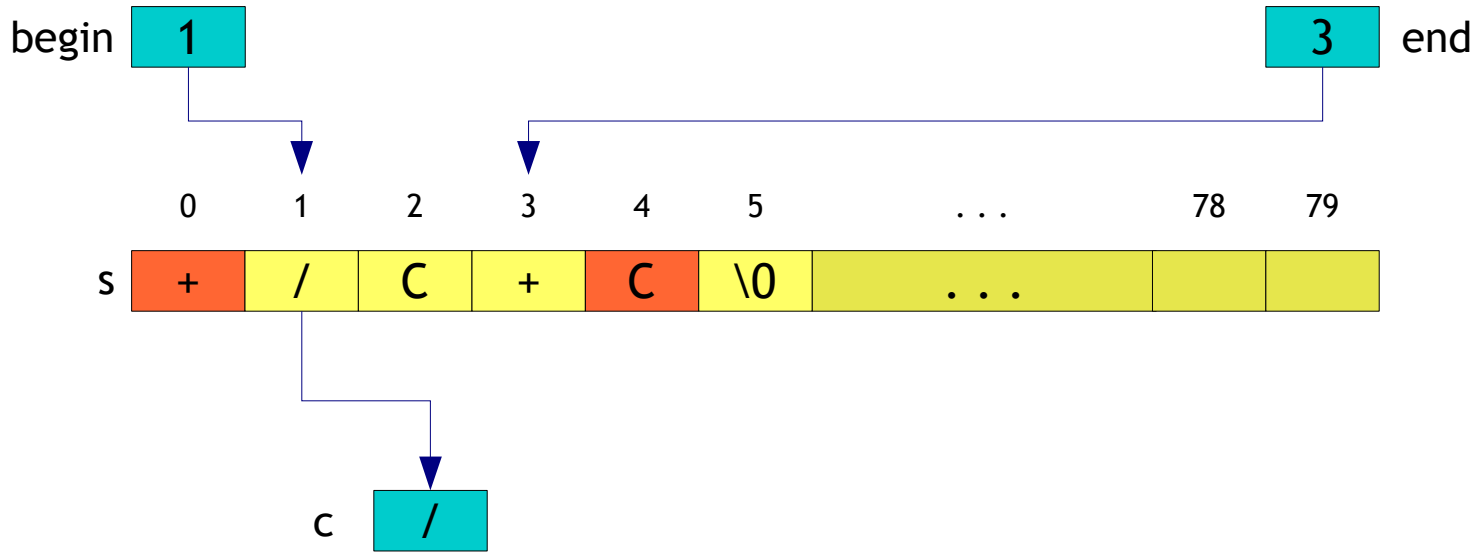
Odwracanie kolejności znaków w napisie — strrev

```
void strrev( char s[] )
{
  int begin, end;
  . . .
  // Zamiana znakow miejscami
  for( begin = 0, end--; begin < end; begin++, end-- )
  {
    char c = s[ begin ];
    s[ begin ] = s[ end ];
    s[ end ] = c;
  }
}
```



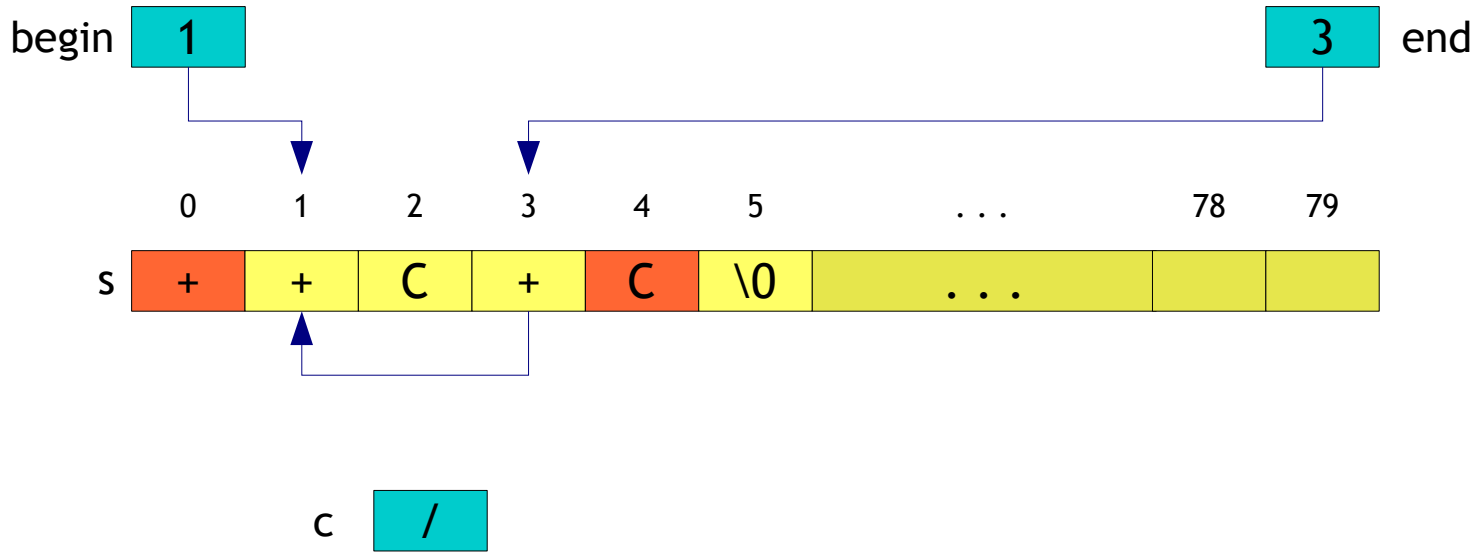
Odwracanie kolejności znaków w napisie — strrev

```
void strrev( char s[] )
{
  int begin, end;
  . . .
  // Zamiana znakow miejscami
  for( begin = 0, end--; begin < end; begin++, end-- )
  {
    char c = s[ begin ];
    s[ begin ] = s[ end ];
    s[ end ] = c;
  }
}
```



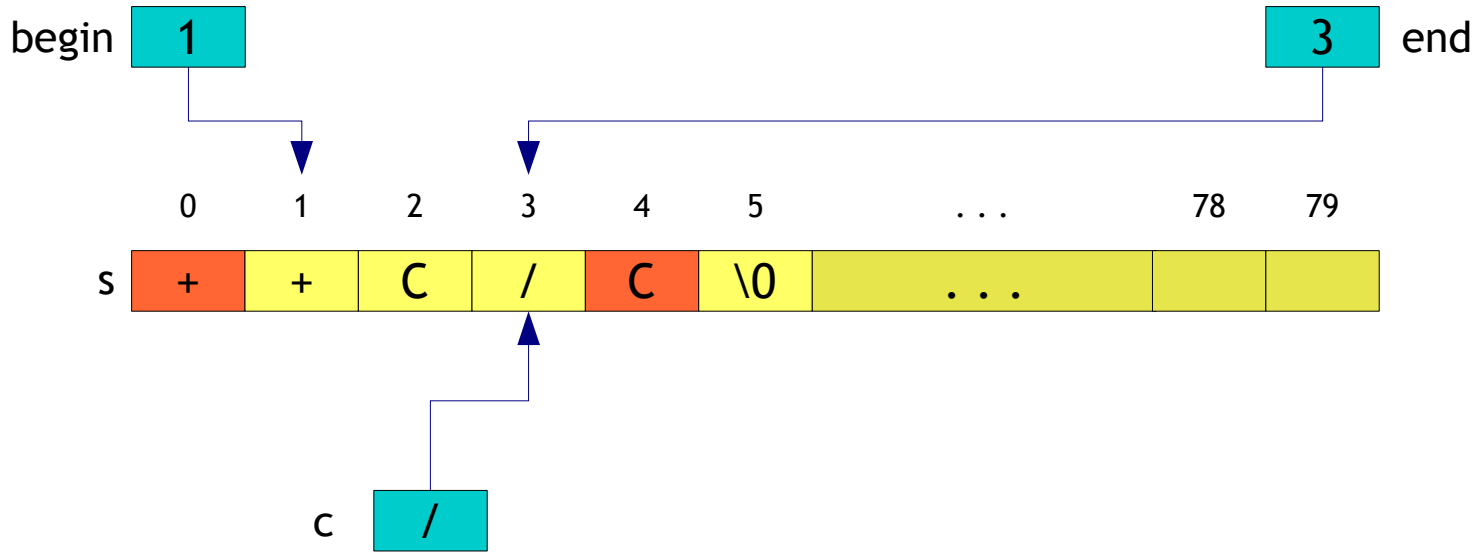
Odwracanie kolejności znaków w napisie — strrev

```
void strrev( char s[] )
{
  int begin, end;
  . . .
  // Zamiana znakow miejscami
  for( begin = 0, end--; begin < end; begin++, end-- )
  {
    char c = s[ begin ];
    s[ begin ] = s[ end ];
    s[ end ] = c;
  }
}
```



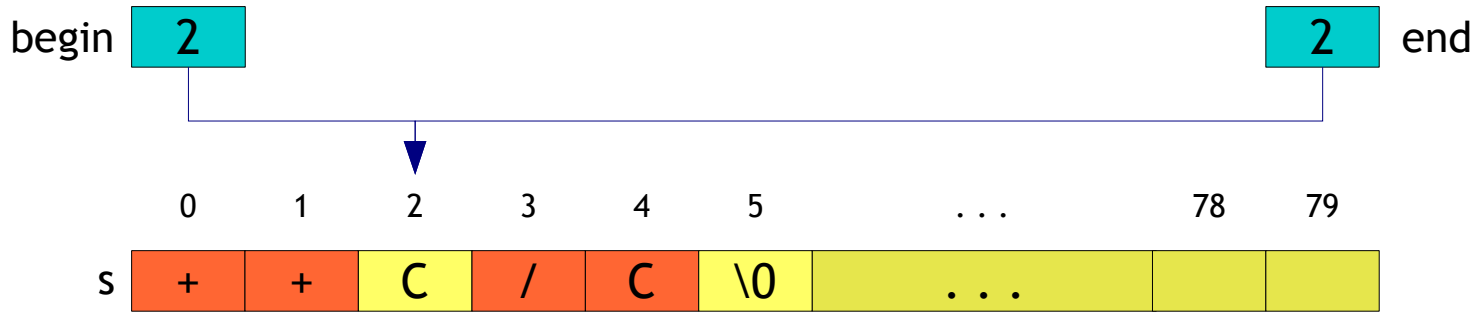
Odwracanie kolejności znaków w napisie — strrev

```
void strrev( char s[] )
{
  int begin, end;
  . . .
  // Zamiana znakow miejscami
  for( begin = 0, end--; begin < end; begin++, end-- )
  {
    char c = s[ begin ];
    s[ begin ] = s[ end ];
    s[ end ] = c;
  }
}
```



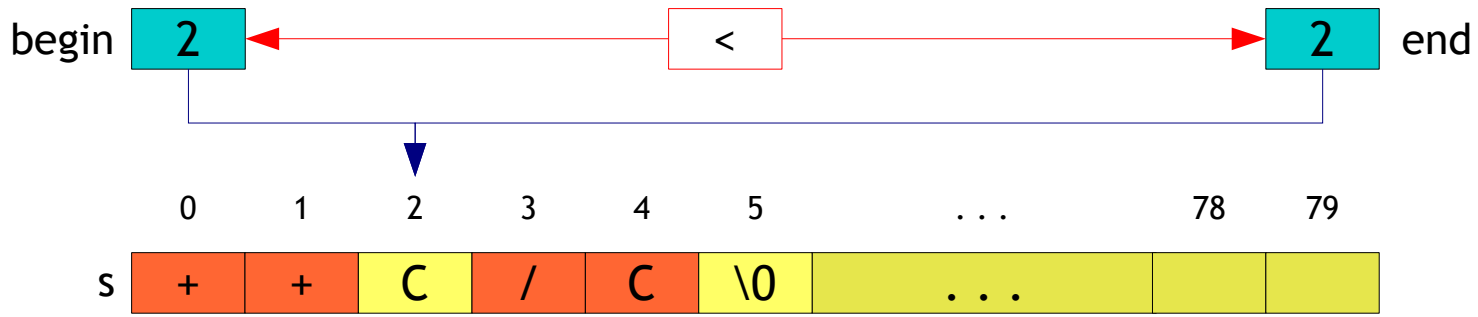
Odwracanie kolejności znaków w napisie — strrev

```
void strrev( char s[] )
{
  int begin, end;
  . . .
  // Zamiana znakow miejscami
  for( begin = 0, end--; begin < end; begin++, end-- )
  {
    char c = s[ begin ];
    s[ begin ] = s[ end ];
    s[ end ] = c;
  }
}
```



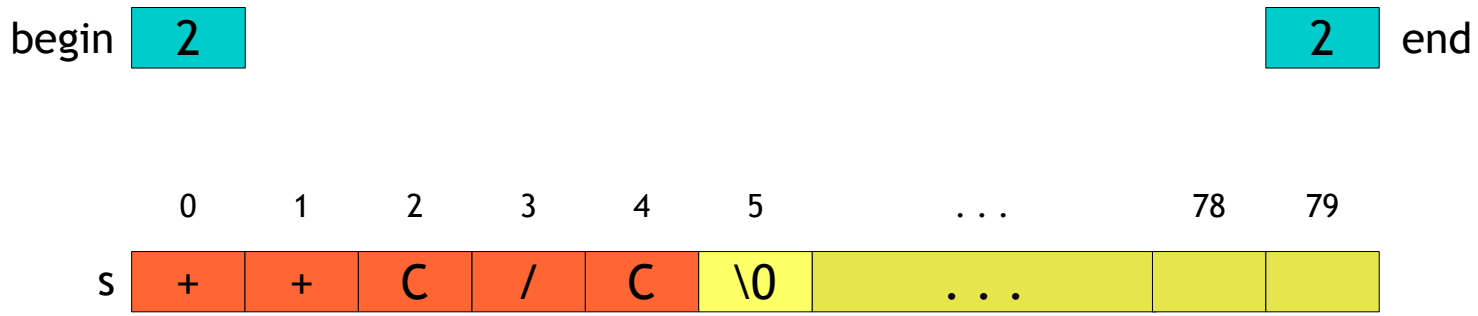
Odwracanie kolejności znaków w napisie — strrev

```
void strrev( char s[] )
{
  int begin, end;
  . . .
  // Zamiana znakow miejscami
  for( begin = 0, end--; begin < end; begin++, end-- )
  {
    char c = s[ begin ];
    s[ begin ] = s[ end ];
    s[ end ] = c;
  }
}
```



Odwracanie kolejności znaków w napisie — strrev

```
void strrev( char s[] )
{
  int begin, end;

  // Szukanie konca napisu
  for( end = 0; s[ end ] != '\0'; end++ )
    ;
  // Zamiana znakow miejscami
  for( begin = 0, end--; begin < end; begin++, end-- )
  {
    char c = s[ begin ];
    s[ begin ] = s[ end ];
    s[ end ] = c;
  }
}
```



To jeszcze nie koniec z tablicami, one będą często wracały, za chwilę powrócą w postaci tablic dynamicznych

Pytania? Polemiki?
Teraz, albo:
roman.siminski@us.edu.pl