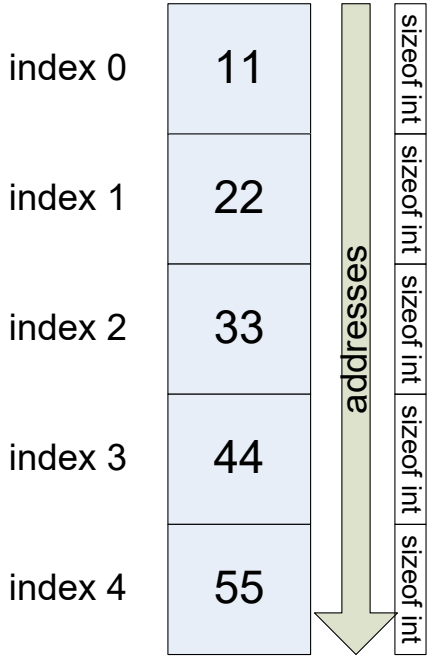
Низови

- Низ у програмском језику Це је тешко одредљива конструкција.
- У суштини представља блок меморије који се тумачи као да се у њему један за другим налазе елементи исте величине
- Кључни аспект низа је његова декларација:

  `element_type array_name[dimension] = {declaration_list};`
- По осталим аспектима је јако сличан показивачу који се не може мењати
  Рецимо, смисао оператора [] је исти

```
int array[5] = {11,22,33,44,55};

printf("element with index 3: %d\n", array[3])
```

Output: 44



Иницијализација низа

- Као и код регуларних променљивих, ако нема експлицитне иницијализације, низови статичке трајности ће бити постављени на нулу (сви њихови елементи), а аутоматске трајности неће.

```
int array[5];
```

local

NDF	NDF	NDF	NDF	NDF

global

0	0	0	0	0

- Листа елемената у витичастим заградам користи се за иницијализацију
- Листа не сме имати више елемената него што је димензија низа. Ако има мање, преостали елементи се попуњавају са нулама.

```
int array[5] = {1,3,5};
```

1	3	5	0	0

- Ако постоји иницијализација низа, димензија може бити изостављена
  Тада ће димензија аутоматски бити једнака величини иницијализаторске листе

```
int array[] = {1, 2, 3}; <=> int array[3] = {1, 2, 3};
```

- У случају да желимо иницијализовати само неке елементе низа, C99 нуди решење:

```
int array[100] = {[13] = 5, [77] = 6};
```

Однос показивача и низова 1/3

- Као што је речено, низови и показивачи су блиски рођаци.

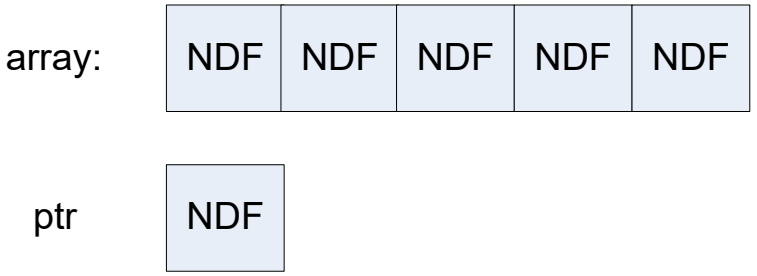
```
int array[] = {1,3,5,7};
int* ptr = array;
if (array[0] == ptr[0]
  && array[1] == ptr[1]
  && array[2] == ptr[2]
  && array[3] == ptr[3])
{
  printf("equal");
}
else
{
  printf("not equal");
}
```

Output: equal

Однос показивача и низова 2/3

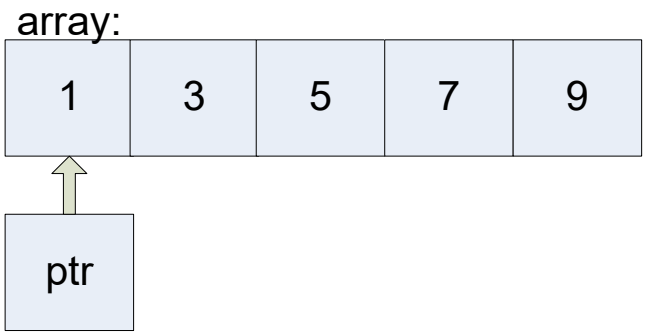
- Али иако су рођаци, нису баш исто
- Када се дефинише низ, заузима се меморија за све његове елементе
- Када се дефинише показивач, заузима се меморија само за њега

```
char array[5];
char* ptr;
```



- Низ, то јест име низа, представља адресу
- Показивач је променљива чија вредност је адреса
- Отприлике, у овом смислу низ је као показивачки литерал

```
int array[] = {1,3,5,7,9};
int* ptr = array;
```



Однос показивача и низова 3/3

- Разлика у резултату sizeof операције
  - За низ враћа број меморијских речи које су заузете за цео низ
  - За показивач - колико меморијских речи треба за адресу

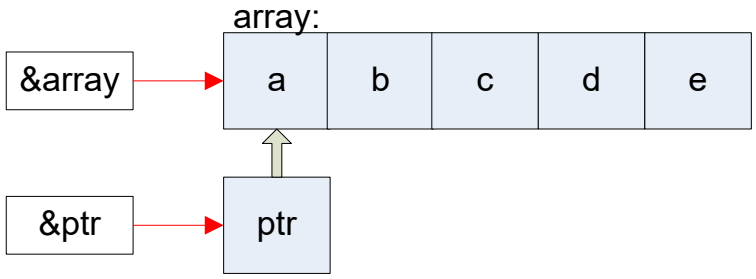
```
char array[15];
char* ptr;

printf("sizeof array: %d\n", sizeof(array));
printf("sizeof pointer: %d\n", sizeof(ptr));
```

Output:
sizeof array: 15
sizeof pointer: 4

- Разлика у & оператору
  - За низ враћа адресу првог елемента
  - За показивач - адресу показивачке променљиве

```
char array[] = {'a','b','c','d','e'};
char* ptr = array;
```

```
array[0] == ptr[0]
&array != &ptr
```



Вишедимензионални низови

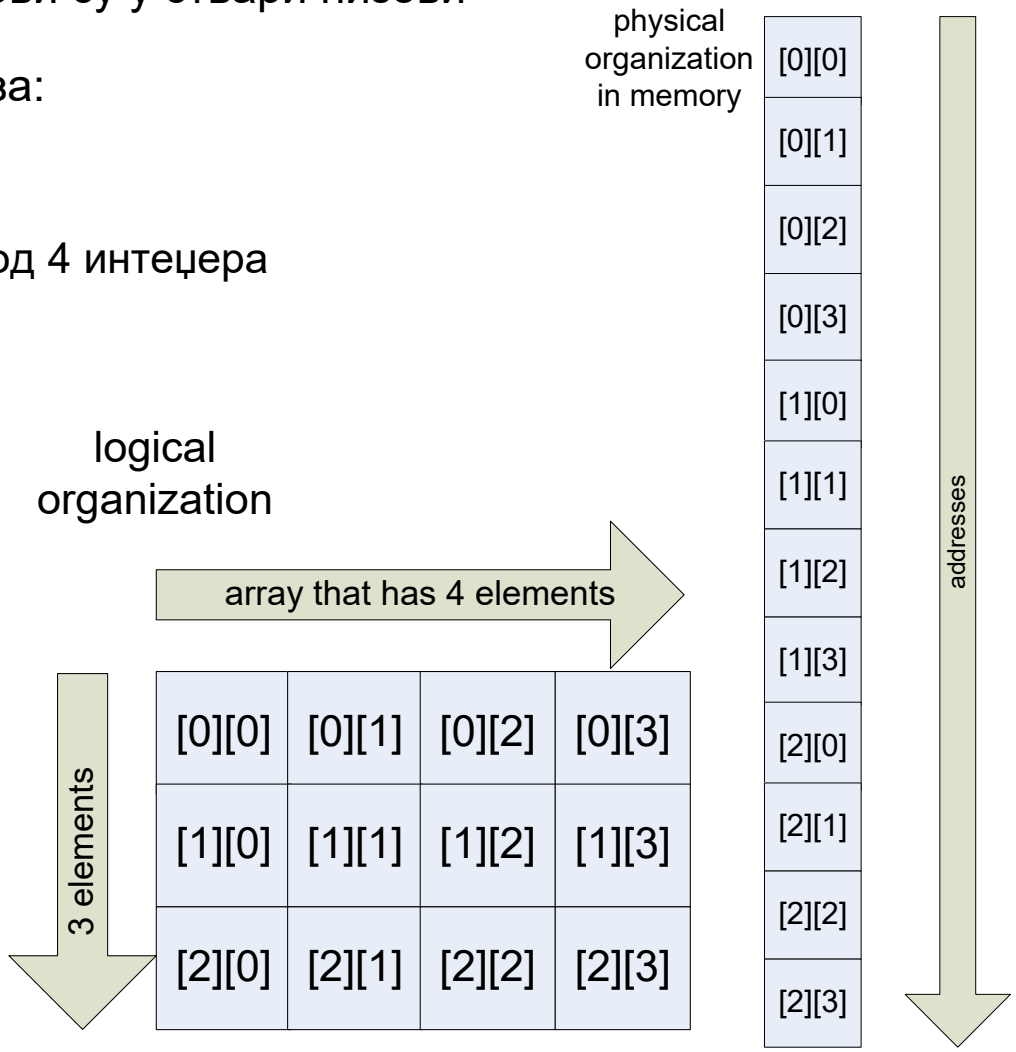
- У Цеу вишедимензионални низови су у ствари низови низова
- Пример дводимензионалног низа:

```
int matrix[3][4];
```

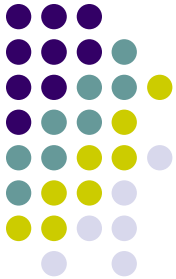
  - то је низ који има 3 елемента
  - а елементи низа су типа низ од 4 интеџера

Еквивалентна дефиниција:

```
typedef int niz4[4];
niz4 matrix[3];

int a;
niz4 y; <=> int y[4];

y[a]
*(y + a*sizeof(int))

int b;
niz4 x[3]; <=> int x[3][4];

x[a][b]
(x[a])[b]
(*(x+a*sizeof(niz4)))[b]
*(x+a*sizeof(niz4)) <=> niz4 T
T[b]
*(T+b*sizeof(int))
```



Прослеђивање низова функцији



- Не може по вредности
- Увек се прослеђује адреса
- Последица је да су наведене декларације практично исте:

```
int func(int arr[10]);
int func(int arr[]);
int func(int* arr);
```

- Број елемената наведен у угластим заградама се игнорише
- За вишедимензионалне низове, димензије морају постојати у свим осим у првим угластим загарадама

```
int func (int arr[][7]);
int func (int* arr[7]);
```

```
element_type name[] [depth1_] … [depth_n]
```

- Зашто?

```
typedef int niz4[4];
niz4 x[3]; <=> int x[3][4];
x[a][b]
*(T + b*sizeof(int)) // T <=> *(x + a*sizeof(niz4))
*(*(x + a*sizeof(niz4)) + b*sizeof(int))
```

Број 4 у декларацији је важан за одређивање sizeof(niz4) - број 3 није важан.

Показивачи на функције 1/3

- Ево како се то декларише

```
return_type (*name)(param_type, param_type);
```

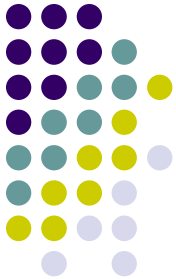
- Слично низовима и имена функција се своде на показивач

```
char* (*fptr) (char* to, const char* from);

fptr = strcpy;  /* OK */
fptr = &strcpy; /* OK */
```

- Позивање функције на коју показивач показује

```
char src[128];
char dst[128];

fptr(dst, src);     /* OK */
(*fptr)(dst, src); /* OK */
```

Показивачи на функције 2/3



- А могу се декларисати и низови показивача на функције

```
return_type (*name[]) (param_type, param_type);
```

```
char* (*fptr[3])(int x) = {func1, func2, func3};

char* pc = fptr[1](7);
char c = *pc;
c = *fptr[1](7); //?
```

```
typedef char* (*fptr_t)(int x);
fptr_t fptr[] = {func1, func2, func3};
```

- Корисно за табеле скокова, или изведбе аутомата са коначним бројем стања

```
int (*fptr[])(int x) = {state1, state2, state3, state4}

int new_state(int current_state, int input)
{
  return fptr[current_state](input);
}
```

Показивачи на функције 3/3

```
int new_state(int current_state, int input)
{
  switch (current_state)
  {
  case 0:
    return state1(input);
    break;
  case 1:
    return state2(input);
    break;
  case 2:
    return state3(input);
    break;
  case 3:
    return state4(input);
    break;
  }
}
```

```
int (*fptr[])(int x) = {state1, state2, state3, state4}

int new_state(int current_state, int input)
{
  return fptr[current_state](input);
}
```