



Низове

гл.ас. д-р. Нора Ангелова

Diagram illustrating the declaration `int const * ptr;`. The code is split into three colored boxes: `int` (green), `const` (orange), and `* ptr;` (blue). A green arrow points from the `int` box to the `const` box. An orange arrow points from the `const` box to the `*` in the `* ptr;` box. A blue arrow points from the `ptr;` part of the `* ptr;` box back to the `*` in the `* ptr;` box. A green line encloses the `int` and `const` boxes, and an orange line encloses the `* ptr;` box.

```
int const * ptr;
```

ptr is a **pointer** to **const int**

Diagram illustrating the declaration `const int * ptr;`. The code is split into three colored boxes: `const` (green), `int` (orange), and `* ptr;` (blue). A green arrow points from the `const` box to the `int` box. An orange arrow points from the `int` box to the `*` in the `* ptr;` box. A blue arrow points from the `ptr;` part of the `* ptr;` box back to the `*` in the `* ptr;` box. A green line encloses the `const` and `int` boxes, and an orange line encloses the `* ptr;` box.

```
const int * ptr;
```

ptr is a **pointer** to **int constant** (i.e. `const int`)

Diagram illustrating the declaration `int * const ptr;`. The code is split into three colored boxes: `int` (green), `*` (orange), and `const ptr;` (blue). A green arrow points from the `int` box to the `*` in the `* const ptr;` box. An orange arrow points from the `*` in the `* const ptr;` box to the `const` in the `const ptr;` box. A blue arrow points from the `ptr;` part of the `const ptr;` box back to the `const` in the `const ptr;` box. A green line encloses the `int` and `*` boxes, and an orange line encloses the `const ptr;` box.

```
int * const ptr;
```

ptr is a **const pointer** to **int**

Diagram illustrating the declaration `const int * const ptr;`. The code is split into three colored boxes: `const int` (orange), `*` (blue), and `const ptr;` (blue). An orange arrow points from the `const int` box to the `*` in the `* const ptr;` box. A blue arrow points from the `ptr;` part of the `const ptr;` box back to the `const` in the `const ptr;` box. An orange line encloses the `const int` and `*` boxes, and a blue line encloses the `const ptr;` box.

```
const int * const ptr;
```

ptr is a **constant pointer** to **const int**

```
int arr[40] = {1, 2, 3, 4};
```

arr -> указател към първи елемент на масива

arr – съдържа адрес на arr[0]

```
arr = &arr[0]
```



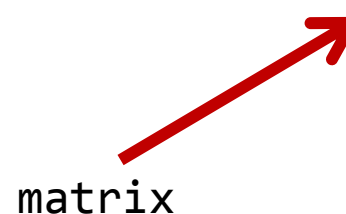
Какъв е типът на arr?

```
int *
```

```
int matrix[5][4] = {  
    { 1,  2,  3,  4},  
    { 5,  6,  7,  8},  
    { 9, 10, 11, 12},  
    {13, 14, 15, 16},  
    {17, 18, 19, 20}  
};
```

matrix -> указател към първи елемент на масива, който също е масив

matrix – съдържа адреса на matrix[0]



1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16
17	18	19	20

```
int matrix[5][4] = {  
    { 1,  2,  3,  4},  
    { 5,  6,  7,  8},  
    { 9, 10, 11, 12},  
    {13, 14, 15, 16},  
    {17, 18, 19, 20}  
};
```

`matrix[i]` -> масив от цели числа

`matrix[i]` - съдържа адреса на `matrix[i][0]`

Какъв е типът на `matrix`?

`<елемент_на_масива>*`

`<масив_от_цели_числа>*`

`int**`

`matrix[i]`



1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16
17	18	19	20

Задача

```
int arr[40] = {1, 2, 3, 4};
```

```
int matrix[5][4] = {  
    { 1,  2,  3,  4},  
    { 5,  6,  7,  8},  
    { 9, 10, 11, 12},  
    {13, 14, 15, 16},  
    {23, 24, 25, 26}  
};
```

Да се изведе на стандартния изход:

1. Адресът на първия елемент на arr;
2. Адресът на последния елемент на arr;
3. Петият елемент на arr;
4. Адресът на matrix;
5. Адресът на третия ред на матрицата matrix;
6. Адресът на елемента на четвърти ред и втори стълб на matrix;
7. Последният елемент на matrix;

Задача

```
int arr[40] = {1, 2, 3, 4};  
int matrix[5][4] = {  
    { 1,  2,  3,  4},  
    { 5,  6,  7,  8},  
    { 9, 10, 11, 12},  
    {13, 14, 15, 16},  
    {23, 24, 25, 26}  
};  
cout << arr;  
cout << arr + 39;  
cout << *(arr + 4) << arr[4];  
cout << matrix;  
cout << matrix + 2;  
cout << *(matrix+3) + 1;  
cout << (*(matrix+4) + 3)  
    << matrix[4][3];
```

Да се изведе на стандартния изход:

1. Адресът на първия елемент на arr;
2. Адресът на последния елемент на arr;
3. Петият елемент на arr;
4. Адресът на matrix;
5. Адресът на третия ред на матрицата matrix;
6. Адресът на елемента на четвърти ред и втори стълб на matrix;
7. Последният елемент на matrix;

Низове

Логическо описание

Крайна или празна редица от символи, заградени в кавички си нарича **СИМВОЛЕН НИЗ**, **ЗНАКОВ НИЗ** или само **НИЗ**.

Пример:

"abc"; "";

Дефиниция

Броят на символите в редицата се нарича **ДЪЛЖИНА** на низа.

Пример:

"abc" - низ с дължина 3

Дефиниция

Низ с дължина 0, се нарича **празен**.

Пример:

"" - празен низ

Низове

Дефиниция

Низ, който се съдържа в даден низ, се нарича негов **подниз**.

Пример:

"a1b2c3"

"b2" - подниз на "a1b2c3"

Дефиниция

Конкатенация на два низа е низ, получен като в края на първия низ се запише вторият.

Пример:

"1*a"; "=a*1";

"1*a=a*1"; - конкатенация на низовете "1*a" и "=a*1"

Низове

Сравнение на низове (лексикографско)

1. Сравнява се всеки символ от първия низ със символа от съответната позиция на втория низ.
2. Сравнението продължава до намиране на два различни символа или до края на поне един от символните низове.

*Ако кодът на символ от първия низ е по-малък от кода на съответния символ от втория низ, или първият низ е изчерпан, приема се, че първият низ е по-малък от втория.

*Ако е по-голям или вторият низ е изчерпан – приема се, че първият низ е по-голям от втория.

*Ако в процеса на сравнение и двата низа едновременно са изчерпани, те са равни

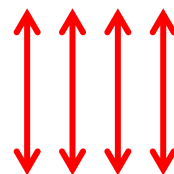
Низове

Сравнение на низове (лексикографско)

Пример:

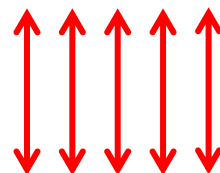
- "abc" == "abc"
- "abc" < "abcd"
- "abc" > "ab"
- "abcd" < "abd"
- "abc" > "aba"

"abcddas" <



"abcdd"

"abcddas" >



"abcdd"

Низове

Физическо представяне

Низовете се представят последователно в паметта



Низове

Представяне на низ:

- като едномерен масив от символи
- като указател към тип `char`

Забележка:

Двата начина са семантично еквиваленти

Низове

(като масиви от символи)

`char str[4];` – определя масив от 4 символа

`char str2[4] = {'a', 'b'};` – определя масив от 4 символа и го инициализира

Забележка:

Дефиницията е еквивалентна на

`char str2[4] = {'a', 'b', '\\0', '\\0'};`

Низове

(като масиви от символи)

`char str2[4] = {'a', 'b'};` – определя масив от 4 символа и го инициализира

Забележка:

Всички действия за работа с масиви са валидни и за работа с масиви от символи.

! Изключение правят операторите за въвеждане и извеждане (>>, <<)

>> - въвежда стойност на променлива от тип масив от СИМВОЛИ

<< - извежда стойност на променлива от тип масив от СИМВОЛИ

Низове

(като масиви от символи)

```
char str2[4] = {'a', 'b'} ;
```

```
cout << str2;
```

няма да изведе адреса на str2[0]

Резултат:

ab

Как се реализира извеждането?

Низове

(като масиви от символи)

```
char str2[4] = {'a', 'b'} ;
```

```
cout << str2;
```

няма да изведе адреса на str2[0]

Резултат:

ab

Забележка:

Ако инициализацията на str2 е пълна (не завършва с '\0')

```
char str2[4] = {'a', 'b', 'c', 'd'};
```

```
cout << str2; // abcd<неопределено>
```

Низове

(като масиви от символи)

Низ – редица от символи, завършваща с ' $\backslash 0$ ';

Предимство:

- Не е необходимо да се пази дължината на низа

Низове

(като масиви от символи)

```
char <променлива>[size] [= “редица_от_символи” |  
{<редица_от_символи>}]опц;
```

- <променлива> - идентификатор
- size

Множество от стойности

- Всички низове с дължина 0, 1, 2, ... size-1

Пример:

(Три еквивалентни дефиниции)

```
char str1[5] = {'a', 'b'};
```

```
char str2[5] = {'a', 'b', '\0', '\0', '\0'};
```

```
char str3[5] = "ab";
```

Низове

(като масиви от символи)

! На променлива от тип низ не може да се присвоява константата от тип низ

```
char str1[5];  
str1 = {'a', 'b'};
```

ИЛИ

```
str1 = "ab";
```

Низове

(като масиви от символи)

Въвеждане на низове:

- Инициализация при декларация
- Индексирани променливи

```
str[0] = 'a';
```

- Чрез операции и вградени функции

Низове

(като масиви от символи)

Операции и вградени функции

Въвеждане на низ:

- `cin`

Пример:

```
char str[5];
```

```
cin >> str;
```

Очаква да се въведе стойност **< 5**.

Знакът за край на низ се добавя автоматично.

Въвеждането продължава до въвеждане на интервал, табулация или знак за нов ред.

Низове

(като масиви от символи)

- `cin.getline(<str>, <size> [, <char>] опц)`

`<str>` - променлива от тип низ

`<size>` - цял израз

`<char>` - произволен символ (по подразбиране `'\n'`)

Въвеждането продължава до срещане на символа `<char>` или до въвеждането на `<size>-1` символа.

Въведеното се записва в `<str>`.

Пример:

```
char str1[5];  
cin.getline(str1, 5);
```

Низове

(като масиви от символи)

Извеждане на низове:

- cout

Пример:

```
char str[5] = "abc";  
cout << str;
```


Низове

(като масиви от символи)

Дължина на низ:

- `strlen(str)`

Пример:

```
char str[5] = "abc";  
cout << strlen(str); // 3
```

Низове

(като масиви от символи)

Конкатенация на низове:

- `strcat(<str1>, <str2>)`

Пример:

```
char str1[7] = "abc";
```

```
char str2[5] = "abc";
```

```
cout << strcat(str1, str2); //abcabc
```

```
    // ! достатъчно място в str1
```

Низове

(като масиви от символи)

Конкатенация на n символа от низ с друг низ:

- `strncat(<str1>, <str2>, n)`
 - Копирането завършва, когато са прехвърлени n символа или е достигнат края на <str2>

Пример:

```
char str1[7] = "abc";
```

```
char str2[5] = "abc";
```

```
cout << strncat(str1, str2, 1); //abca  
// ! достатъчно място в str1
```

Низове

(като масиви от символи)

Сравняване на низове:

- `strcmp(<str1>, <str2>)`

Връща: **-1** | **0** | **1**

Пример:

```
char str1[7] = "abc";
```

```
char str2[4] = "abc";
```

```
cout << strcmp(str1, str2); // 0
```

Низове

(като масиви от символи)

Сравняване на n символа на низове:

• `strncmp(<str1>, <str2>, n)`

Връща: **-1** | **0** | **1**

- Ако n е по-малко от дължината на <str1>, се сравняват <str1>, <str2>

Пример:

```
char str1[7] = "abc";
```

```
char str2[4] = "abc";
```

```
cout << strncmp(str1, str2, 2); // 0
```

Низове

(като масиви от символи)

Копиране на низове:

- `strcpy(<str1>, <str2>)`

// ! достатъчно място в str1

Пример:

```
char str1[10];
```

```
char str2[4] = "abc";
```

```
strcpy(str1, str2);
```

```
cout << str1;
```

Низове

(като масиви от символи)

Копиране на n символа на низове:

- `strncpy(<str1>, <str2>, n)`
// ! достатъчно място в `str1`

Пример:

```
char str1[10];
```

```
char str2[4] = "abc";
```

```
strncpy(str1, str2, 2);
```

```
cout << str1; // ab
```

```
strncpy(str1, str2, 5); // 5 > strlen(str2), допълва се с '\0'
```

```
cout << str1; // abc
```

Низове

(като масиви от символи)

Копиране на n символа на низове:

- `strncpy(<str1>, <str2>, n)`
 // ! достатъчно място в str1

Пример:

```
char str1[10];
```

```
char str2[4] = "abc";
```

```
strncpy(str1, str2, 12);
```

```
cout << str1; // error; 12 > 10
```


Низове

(като масиви от символи)

Търсене на низ в друг низ:

• `strstr(<str1>, <str2>)`

Връща:

- Подниза на <str1>, започващ от първото срещане на <str2> до края на <str1>.
- NULL

Пример:

```
char str1[10] = "1234abcdf";
```

```
char str2[5] = "ab";
```

```
cout << strstr(str1, str2); // abcdf
```

Низове

(като масиви от символи)

Пример:

```
char str1[10] = "1234abcdf";
```

```
char str2[5] = "567";
```

```
cout << strstr(str1, str2); // грешка
```

```
if (strstr(str1, str2)) {...} // false
```

Низове

(като масиви от символи)

Търсене на символ в низ:

- `strchr(<str1>, <expr>)`
 - `<expr>` - израз от интегрален или изброен тип с положителна стойност, означаваща ASCII код на СИМВОЛ

Връща:

- Подниза на `<str1>`, започващ от първото срещане на символ с ASCII `<expr>` до края на `<str1>`.
- `NULL`

Пример:

```
char str1[10] = "abcdf";  
cout << strchr(str1, 'd'); // df
```

Низове

(като масиви от символи)

Пример:

```
char str1[10] = "1234abcdf";
```

```
cout << strchr(str1, '5'); // грешка
```

```
if (strchr(str1, '5')) {...} // false
```

Низове

(като масиви от символи)

Търсене на първа разлика:

- `strspn(<str1>, <str2>)`

Връща:

- Дължината на низа до първия различен символ

Пример:

```
char str1[6] = "abcds";
```

```
char str2[6] = "abdse";
```

```
cout << strspn(str1, str2) << endl; // 2
```

Низове

(като масиви от символи)

Преобразуване на низ в цяло число:

• `atoi(<str>)`

Връща:

- Преобразува низа `<str>` в число от тип `int`.
- Сканира се до първия символ различен от символ.
- Интервалите, табулациите и `'\n'` се пренебрегват.

Пример:

```
char str1[10] = "-1234abcdf";  
cout << atoi(str1); // -1234
```

Низове

(като масиви от символи)

Преобразуване на низ в цяло число:

- `atoi(<str>)`

Пример:

```
char str2[10] = "a123bcdf";  
cout << atoi(str1); // 0
```

Низове

(като масиви от символи)

Преобразуване на низ в реално число:

• `atof(<str>)`

Връща:

- Преобразува низа <str> в число от тип double.
- Сканира се до първия символ различен от символ, който не е '.'.
- Интервалите, табулациите и '\n' се пренебрегват.

Пример:

```
char str1[10] = "-123.4abcdf";  
cout << atof(str1); // -123.4
```


Низове

(като масиви от символи)

Преобразуване на низ в реално число:

- `atof(<str>)`

Пример:

```
char str1[10] = ".4abcdf";  
cout << atof(str1); // 0.4
```

```
char str2[10] = "abcdf";  
cout << atof(str2); // 0
```

Низове

(като масиви от символи)

Задача.

Даден е масив от цели числа с размерност 20.

- Да се провери има ли поне два последователни елемента с равни стойности.

Дадена е матрица от цели числа с размери $n \times m$.

- Да се намери сумата на елементите над главния диагонал.



```
cout << “Край”;
```