

23.09.2024

# Строки. Символьный ввод-вывод и достоверность ввода. Массивы. Указатели.

*Филиппов Михаил Витальевич*

[m.filippov@g.nsu.ru](mailto:m.filippov@g.nsu.ru)

89232283872

Императивное программирование, 2024-2025

**N** \* Новосибирский  
государственный  
университет  
**\*НАСТОЯЩАЯ НАУКА**

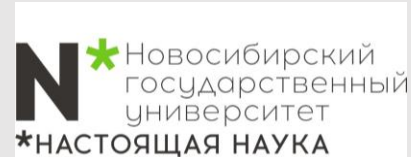


# Давайте познакомимся



## Филиппов Михаил Витальевич

- Окончил магистратуру ФФ НГУ
- Окончил аспирантуру ИТ СО РАН
- Являюсь м.н.с. ИТ СО РАН
- 7+ лет опыт в программировании C/C++



# Адженда

**Строки**

**30 минут**

**Символьный  
ввод-вывод и  
достоверность  
ввода**

**25 минут**

**Массивы.  
Указатели.**

**35 минут**

# Адженда

**Строки**

**30 минут**

**Символьный  
ввод-вывод и  
достоверность  
ввода**

**25 минут**

**Массивы.  
Указатели.**

**35 минут**

# Демонстрационная программа

```
#include <stdio.h>
#include <string.h> // для прототипа функции strlen()
#define DENSITY 62.4 // удельная масса человека в фунтах на кубический фут
int main()
{
    float weight, volume;
    int size, letters;
    char name[40]; // name представляет собой массив из 40 символов
    printf("Здравствуйте! Как вас зовут?\n");
    scanf("%s", name);
    printf("%s, сколько вы весите в фунтах?\n", name);
    scanf("%f", &weight);
    size = sizeof name;
    letters = strlen(name);
    volume = weight / DENSITY;
    printf("Хорошо, %s, ваш объем составляет %2.2f кубических футов.\n", name, volume);
    printf("К тому же ваше имя состоит из %d букв, ", letters);
    printf("и мы располагаем 40 байтами для его сохранения.\n", size);
    return 0;
}

// Здравствуйте! Как вас зовут?
// Mikhail
// Mikhail, сколько вы весите в фунтах?
// 195
// Хорошо, Mikhail, ваш объем составляет 3.12 кубических футов.
// К тому же ваше имя состоит из 7 букв, и мы располагаем 40 байтами для его сохранения.
```

# Массив типа `char` и нулевой символ



Каждая ячейка содержит один байт

Нулевой символ

**Символьная строка** — это последовательность из одного или большего количества символов

**Пример:**

"Это длинная строка символов."

Двойные кавычки не являются частью строки. Они сообщают компилятору, что внутри них содержится строка точно так же, как одиночные кавычки идентифицируют символ.

На последней позиции массива находится символ `\0`. Он представляет собой **нулевой символ**, который в языке C служит **для пометки конца строки**.

**Нулевой символ** — это не цифра ноль, а непечатаемый символ, кодовое значение которого в кодировке ASCII (или эквивалентной) равно 0. Строки в C всегда сохраняются с завершающим нулевым символом.




# Использование строк

```
#include <stdio.h>
#include <string.h> // для прототипа функции strlen()
#define PRAISE "Вы - выдающаяся личность."
int main()
{
    system("chcp 65001");
    char name[40];
    printf("Как вас зовут? ");
    scanf("%s", name);
    printf("Здравствуйте, %s. %s\n", name, PRAISE);
    return 0;
}
// Как вас зовут? Mikhail
// Здравствуйте, Mikhail. Вы - выдающаяся личность.
```

Спецификатор %s сообщает функции printf () о необходимости вывода строки.



# Различия между строками и символами

Символ 'x' 

Строка "x" 

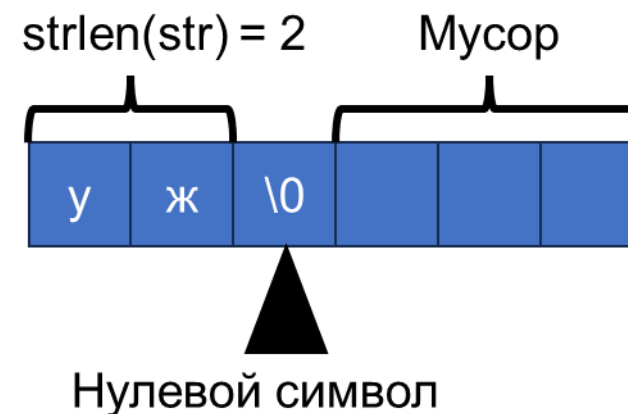
Строковая константа "x" — вовсе **не то же самое, что и** символьная константа 'x'. Одно из различий связано с тем, что 'x' имеет базовый тип (char), но "x" — это производный тип, представляющий собой массив значений char. Второе различие заключается в том, что "x" на самом деле состоит из двух символов — 'x' и '\0'.





# Функция strlen()

```
#include <stdio.h>
#include <string.h> // для прототипа функции strlen()
#define PRAISE "Вы - выдающаяся личность."
int main()
{
    system("chcp 65001");
    char name[40];
    printf("Как вас зовут? ");
    scanf("%s", name);
    printf("Здравствуйте, %s. %s\n", name, PRAISE);
    return 0;
}
// Как вас зовут? Mikhail
// Здравствуйте, Mikhail. Вы - выдающаяся личность.
```



Спецификатор %s сообщает функции printf () о необходимости вывода строки. м

# Некоторые символические константы из файла `limits.h`

| Символическая константа | Что представляет                                       |
|-------------------------|--|
| <code>CHAR_BIT</code>   | Количество битов в типе <code>char</code>              |
| <code>CHAR_MAX</code>   | Максимальное значение типа <code>char</code>           |
| <code>CHAR_MIN</code>   | Минимальное значение типа <code>char</code>            |
| <code>SCHAR_MAX</code>  | Максимальное значение типа <code>signed char</code>    |
| <code>SCHAR_MIN</code>  | Минимальное значение типа <code>signed char</code>     |
| <code>UCHAR_MAX</code>  | Максимальное значение типа <code>unsigned char</code>  |
| <code>SHRT_MAX</code>   | Максимальное значение типа <code>short</code>          |
| <code>SHRT_MIN</code>   | Минимальное значение типа <code>short</code>           |
| <code>USHRT_MAX</code>  | Максимальное значение типа <code>unsigned short</code> |



# Некоторые символические константы из файла `limits.h`

| Символическая константа | Что представляет   |
|-------------------------|--|
| <code>INT_MAX</code>    | Максимальное значение типа <code>int</code>                |
| <code>INT_MIN</code>    | Минимальное значение типа <code>int</code>                 |
| <code>UINT_MAX</code>   | Максимальное значение типа <code>unsigned int</code>       |
| <code>LONG_MAX</code>   | Максимальное значение типа <code>long</code>               |
| <code>LONG_MIN</code>   | Минимальное значение типа <code>long</code>                |
| <code>ULONG_MAX</code>  | Максимальное значение типа <code>unsigned long</code>      |
| <code>LLONG_MAX</code>  | Максимальное значение типа <code>long long</code>          |
| <code>LLONG_MIN</code>  | Минимальное значение типа <code>long long</code>           |
| <code>ULLONG_MAX</code> | Максимальное значение типа <code>unsigned long long</code> |



# Некоторые символические константы из файла float.h

| Символическая константа | Что представляет  |
|-------------------------|---|
| FLT_MANT_DIG            | Количество битов в мантиссе типа float  |
| FLT_DIG                 | Минимальное количество значащих десятичных цифр для типа float  |
| FLT_MIN_10_EXP          | Минимальное значение отрицательного десятичного порядка для типа float с полным набором значащих цифр |
| FLT_MAX_10_EXP          | Максимальное значение положительного десятичного порядка для типа float                               |
| FLT_MIN                 | Минимальное значение для положительного числа типа float, сохраняющего полную точность                |
| FLT_MAX                 | Максимальное значение для положительного числа типа float   |
| FLT_EPSILON             | Разница между 1.00 и минимальным значением float, которое больше 1.00                                 |



# Использование данных float.h и limits.h.

```
#include <stdio.h>
#include <limits.h> // пределы для целых чисел
#include <float.h>  // пределы для чисел с плавающей запятой
int main()
{
    system("chcp 65001");
    printf("Некоторые пределы чисел для данной системы:\n");
    printf("Наибольшее значение типа int: %d\n", INT_MAX);
    printf("Наименьшее значение типа long long: %lld\n", LLONG_MIN);
    printf("В данной системе один байт = %d битов.\n", CHAR_BIT);
    printf("Наибольшее значение типа double: %e\n", DBL_MAX);
    printf("Наименьшее нормализованное значение типа float: %e\n", FLT_MIN);
    printf("Точность значений типа float = %d знаков\n", FLT_DIG);
    printf("Разница между 1.00 и минимальным значением float, которое больше 1.00 = %e\n", FLT_EPSILON);
    return 0;
}
// Некоторые пределы чисел для данной системы:
// Наибольшее значение типа int: 2147483647
// Наименьшее значение типа long long: -9223372036854775808
// В данной системе один байт = 8 битов.
// Наибольшее значение типа double: 1.797693e+308
// Наименьшее нормализованное значение типа float: 1.175494e-38
// Точность значений типа float = 6 знаков
// Разница между 1.00 и минимальным значением float, которое больше 1.00 = 1.192093e-07
```



# printf() - спецификаторы преобразования

| Спецификатор преобразования | Описание вывода   |
|-----------------------------|---|
| %a                          | Число с плавающей запятой, шестнадцатеричные цифры и p-запись (C99/C11)   |
| %A                          | Число с плавающей запятой, шестнадцатеричные цифры и P-запись (C99/C11)   |
| %c                          | Одиночный символ  |
| %d                          | Десятичное целое число со знаком  |
| %e                          | Число с плавающей запятой, экспоненциальное представление   |
| %E                          | Число с плавающей запятой, экспоненциальное представление   |
| %f                          | Число с плавающей запятой, десятичное представление   |
| %g                          | В зависимости от значения использует %f или %e. Спецификатор %e применяется, если показатель степени меньше -4 либо больше или равен указанной точности |



# printf() - спецификаторы преобразования

| Спецификатор преобразования | Описание вывода   |
|-----------------------------|---|
| %G                          | В зависимости от значения использует %f или %E. Спецификатор %E применяется, если показатель степени меньше -4 либо больше или равен указанной точности |
| %i                          | Десятичное целое число со знаком (то же, что и %d)  |
| %o                          | Восьмеричное целое число без знака  |
| %p                          | Указатель   |
| %s                          | Символьная строка   |
| %u                          | Десятичное целое число без знака  |
| %x                          | Шестнадцатеричное целое число без знака, используются шестнадцатеричные цифры 0f  |
| %X                          | Шестнадцатеричное целое число без знака, используются шестнадцатеричные цифры 0F  |
| %%                          | Знак процента   |

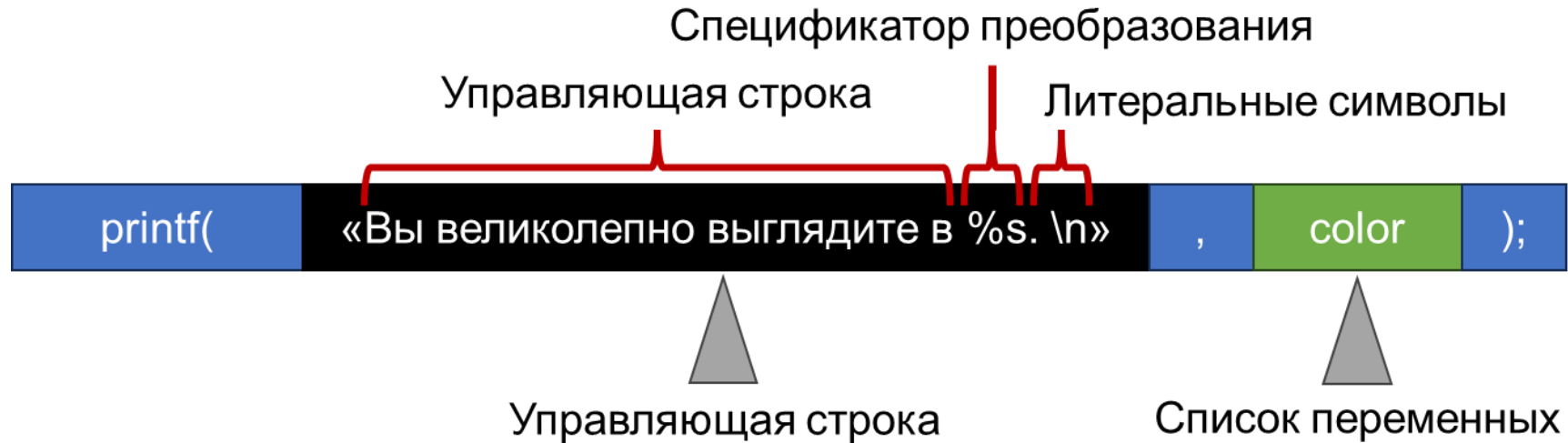


# printf() - демо

```
#include <stdio.h>
#define PI 3.141593
int main()
{
    system("chcp 65001");
    int number = 7;
    float pies = 12.75;
    int cost = 7800;
    printf("%d участников соревнований съели %f пирожков с  
вишнями.\n", number, pies);
    printf("Значение pi равно %f.\n", PI);
    printf("До свидания! Ваше искусство слишком дорого  
обходится,\n");
    printf("%c%d\n", '$', 2 * cost);
    return 0;
}
// 7 участников соревнований съели 12.750000 пирожков с вишнями.
// Значение pi равно 3.141593.
// До свидания! Ваше искусство слишком дорого обходится,
// $15600
```



# printf() - демо



Управляющая строка содержит два разных вида информации:

- символы, которые в действительности выводятся;
- спецификаторы преобразования.



# Модификаторы функции printf()

| Модификатор    | Описание  |
|----------------|---|
| флаг           | Пять допустимых флагов (-, +, пробел, # и 0). Можно указывать ноль или больше флагов.   |
| цифра(цифры)   | Минимальная ширина поля. Если выводимое число или строка не умецаются в это поле, будет использоваться поле большей ширины. Пример: "%4d"   |
| .цифра(.цифры) | Точность. Для преобразований %e, %E и %f указывается количество цифр, которые будут выведены справа от десятичной точки. Для преобразований %g и %G задается максимальное количество значащих цифр. Для преобразований %s определяется максимальное количество символов, которое может быть выведено. Для целочисленных преобразований указывается минимальное количество отображаемых цифр; при необходимости для соответствия с этим минимумом применяются ведущие нули. Использование только точки (.) предполагает, что далее следует ноль, т.е %f — то же самое, что и %.0f. Пример: "%5.2f" выводит значение типа float в поле шириной пять символов и двумя цифрами после десятичной точки |
| h              | Используется со спецификатором целочисленного преобразования для отображения значений типа short int или unsigned short int. Примеры: "%hu", "%hx" и "%6.4hd".  |
| hh             | Используется со спецификатором целочисленного преобразования для отображения значений типа signed char или unsigned char. Примеры: "%hhu", "%hhx" и "%6.4hhd"   |





# Модификаторы функции printf()

| Модификатор | Описание  |
|-------------|---|
| j           | Используется со спецификатором целочисленного преобразования для отображения значений <code>intmax_t</code> или <code>uintmax_t</code> ; эти типы определены в <code>stdint.h</code> . Примеры: <code>"%jd"</code> и <code>"%.8jX"</code> |
| l           | Используется со спецификатором целочисленного преобразования для отображения значений типа <code>long int</code> или <code>unsigned long int</code> . Примеры: <code>"%ld"</code> и <code>"i8lu"</code>                                   |
| ll          | Используется со спецификатором целочисленного преобразования для отображения значений типа <code>long long int</code> или <code>unsigned long long int</code> (стандарт C99). Примеры: <code>"%lld"</code> и <code>"%8llu"</code>         |
| L           | Используется со спецификатором преобразования значений с плавающей запятой для отображения значений типа <code>long double</code> . Примеры: <code>"%Lf"</code> и <code>"10.4Le"</code>   |
| t           | Используется со спецификатором целочисленного преобразования для отображения значений <code>ptrdiff_t</code> . Этот тип соответствует разницы между двумя указателями (стандарт C99). Примеры: <code>"%td"</code> и <code>"%12ti"</code>  |
| z           | Используется со спецификатором целочисленного преобразования для отображения значений <code>size_t</code> . Этот тип возвращается операцией <code>sizeof</code> (стандарт C99). Примеры: <code>"%zd"</code> и <code>"%12zx"</code>        |



# Флаги функции printf()

| Модификатор | Описание  |
|-------------|---|
| j           | Используется со спецификатором целочисленного преобразования для отображения значений <code>intmax_t</code> или <code>uintmax_t</code> ; эти типы определены в <code>stdint.h</code> . Примеры: <code>"%jd"</code> и <code>"%.8jX"</code> |
| l           | Используется со спецификатором целочисленного преобразования для отображения значений типа <code>long int</code> или <code>unsigned long int</code> . Примеры: <code>"%ld"</code> и <code>"i8lu"</code>                                   |
| ll          | Используется со спецификатором целочисленного преобразования для отображения значений типа <code>long long int</code> или <code>unsigned long long int</code> (стандарт C99). Примеры: <code>"%lld"</code> и <code>"%8llu"</code>         |
| L           | Используется со спецификатором преобразования значений с плавающей запятой для отображения значений типа <code>long double</code> . Примеры: <code>"%Lf"</code> и <code>"10.4Le"</code>   |
| t           | Используется со спецификатором целочисленного преобразования для отображения значений <code>ptrdiff_t</code> . Этот тип соответствует разницы между двумя указателями (стандарт C99). Примеры: <code>"%td"</code> и <code>"%12ti"</code>  |
| z           | Используется со спецификатором целочисленного преобразования для отображения значений <code>size_t</code> . Этот тип возвращается операцией <code>sizeof</code> (стандарт C99). Примеры: <code>"%zd"</code> и <code>"%12zx"</code>        |



# Примеры использования модификаторов и флагов

```
#include <stdio.h>
#define PAGES 959
int main()
{
    printf("%d*\n", PAGES);
    printf("%2d*\n", PAGES);
    printf("%10d*\n", PAGES);
    printf("%-10d*\n", PAGES);
    return 0;
}
// *959*
// *959*
// *          959*
// *959          *
```



# Примеры использования модификаторов и флагов

```
#include <stdio.h>
int main()
{
    const double RENT = 3852.99;
    // константа, объявленная
    // посредством const
    printf("%f*\n", RENT);
    printf("%e*\n", RENT);
    printf("%.2f*\n", RENT);
    printf("%.1f*\n", RENT);
    printf("%.3f*\n", RENT);
    printf("%.3E*\n", RENT);
    printf("%.4f*\n", RENT);
    printf("%.0f*\n", RENT);
    return 0;
}
```

```
// *3852.990000*
// *3.852990e+03*
// *3852.99*
// *3853*
// * 3852.990*
// * 3.853E+03*
// *+3852.99*
// *0003852.99*
```



# Примеры использования модификаторов и флагов

```
#include <stdio.h>
#define PAGES 959
int main()
{
    printf("%x %X %#x\n", 31, 31, 31);
    printf("***%d**% d**% d**\n", 42, 42, -42);
    printf("***%5d**%5.3d**%05d**%05.3d**\n", 6, 6, 6, 6);
    return 0;
}
// 1f 1F 0x1f
// **42** 42**-42**
// **      6**  006**00006**  006**
```





# Примеры использования модификаторов и флагов

```
#include <stdio.h>
#define BLURB "Authentic imitation!"
int main()
{
    printf("[%2s]\n", BLURB);
    printf("[%24s]\n", BLURB);
    printf("[%24.5s]\n", BLURB);
    printf("[% -24.5s]\n ", BLURB);
    return 0;
}
// [Authentic imitation!]
// [      Authentic imitation!]
// [                          Authe]
// [Authe                      ]
```



# Несовпадающие преобразования

```
#include <stdio.h>
#define PAGES 336
#define WORDS 65618
int main()
{
    short num = PAGES;
    short mnum = -PAGES;
    printf("num как тип short и тип unsigned short: %hd %hu\n", num, num);
    printf("-num как тип short и тип unsigned short: %hd %hu\n", mnum, mnum);
    printf("num как тип int и тип char: %d %c\n", num, num);
    printf("WORDS как тип int, short и char: %d %hd %c \n", WORDS, WORDS, WORDS);
    return 0;
}
// num как тип short и тип unsigned short: 336 336
// -num как тип short и тип unsigned short: -336 65200
// num как тип int и тип char: 336 P
// WORDS как тип int, short и char: 65618 82 Rc
```

# Вывод длинных строк

```
#include <stdio.h>
int main()
{
    printf("Вот один из способов вывода ");
    printf("длинной строки.\n");
    printf("Вот второй способ вывода \
длинной строки.\n");
    printf("А вот самый новый способ вывода "
           "длинной строки.\n"); /* ANSI C */
    return 0;
}
// Вот один из способов вывода длинной строки.
// Вот второй способ вывода длинной строки.
// А вот самый новый способ вывода длинной строки.
```



# Использование функции scanf()

```
#include <stdio.h>
int main()
{
    int age;          // переменная
    float assets;     // переменная
    char pet[30];     // строка
    printf("Введите информацию о своем возрасте, сумме в банке и любимом животном.\n");
    scanf("%d %f", &age, &assets); // здесь должен быть указан символ &
    scanf("%s", pet);           // для строкового массива символ & не нужен
    printf("%d $%.2f %s\n", age, assets, pet);
    return 0;
}
// Введите информацию о своем возрасте, сумме в банке и любимом животном.
// 27 10000000 Cat
// 27 $10000000.00 Cat
```

# Спецификаторы преобразования ANSI C для функции scanf()

| Спецификатор преобразования | Значение   |
|-----------------------------|--|
| %c                          | Интерпретирует введенные данные как символ   |
| %d                          | Интерпретирует введенные данные как десятичное целое число со знаком   |
| %e, %f, %g, %a              | Интерпретирует введенные данные как число с плавающей запятой (%a появился в C99)  |
| %E, %F, %G, %A              | Интерпретирует введенные данные как число с плавающей запятой (%A появился в C99)  |
| %i                          | Интерпретирует введенные данные как десятичное целое число со знаком   |
| %o                          | Интерпретирует введенные данные как восьмеричное целое число со знаком   |
| %P                          | Интерпретирует введенные данные как указатель (адрес)  |
| %s                          | Интерпретирует введенные данные как строку. Ввод начинается с первого символа, не являющегося пробельным, и включает все символы до следующего пробельного символа |
| %u                          | Интерпретирует введенные данные как десятичное целое число без знака   |
| %x, %X                      | Интерпретирует введенные данные как шестнадцатеричное целое число со знаком  |



# Модификаторы преобразования функции scanf()

| Модификатор  | Значение   |
|--------------|--|
| *            | Подавляет присваивание. Пример: "%*d".   |
| цифра(цифры) | Максимальная ширина поля. Ввод прекращается, когда достигнута максимальная ширина поля или если обнаружен первый пробельный символ (в зависимости от того, что произойдет раньше). Пример: "%10s"  |
| hh           | Читает целое число как signed char или unsigned char. Примеры: "%hhd", "%hhu"  |
| ll           | Читает целочисленное число как long long или как long long без знака (стандарт C99). Примеры: "%lld", "%llu"   |
| h, l или L   | Спецификаторы "%hd" и "%hi" указывают, что значение будет сохранено с типом short int. Спецификаторы "%ho", "%hx" и "%hu" определяют, что значение будет сохранено с типом unsigned short int. Спецификаторы "%ld" и "%li" указывают, что значение будет сохранено с типом long. Спецификаторы "%lo", "%lx" и "%lu" определяют, что значение будет сохранено с типом unsigned long. Спецификаторы "%le", "%lf" и "%lg" указывают, что значение будет сохранено с типом double. Использование модификатора L вместо l в сочетании с e, f и g определяет, что значение будет сохранено с типом long double. В отсутствие этих модификаторов d, i, o их указывают на тип int, а e, f и g — на тип float |

# Модификаторы преобразования функции scanf()

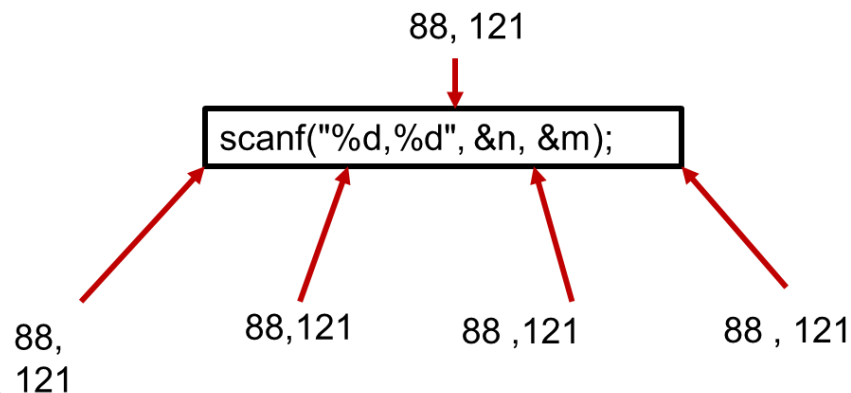
| Модификатор | Значение   |
|-------------|--|
| j           | Когда за ним следует спецификатор целочисленного преобразования, указывает на использование типа <code>intmax_t</code> или <code>uintmax_t</code> (стандарт C99). Примеры: "%jd" "%ju"               |
| z           | Когда за ним следует спецификатор целочисленного преобразования, указывает на использование типа, возвращенного операцией <code>sizeof</code> (стандарт C99). Примеры: "%zd" "%zo"                   |
| t           | Когда за ним следует спецификатор целочисленного преобразования, указывает на использование типа, служащего для представления разницы между двумя указателями (стандарт C99). Примеры: "%td", "%tx". |

# Обработка ввода функцией scanf()

*Предположим, что вы применяете спецификатор %d, чтобы прочесть целое число.*

- Функция scanf() начинает читать поток ввода по одному символу за раз.
- Она пропускает пробельные символы пока не натолкнется на символ, отличный от пробельного.
- Поскольку функция scanf() пытается прочесть целое число, она ожидает обнаружить цифровой символ или, возможно, знак (+ или -).
- Встретив цифру или знак, она запоминает этот символ и считывает следующий, пока не столкнется с нецифровым символом.
- Функция scanf() помещает этот нецифровой символ обратно в поток ввода.
- Наконец, функция scanf() вычисляет числовое значение, соответствующее считанным ею цифрам (и, возможно, знаку), и заносит это значение в указанную переменную.

Если вы используете ширину поля, функция scanf() прекращает чтение при достижении конца поля или на первом пробельном символе, в зависимости от того, что произойдет раньше.



# Использование функции scanf() - демо

```
#include <stdio.h>
int main()
{
    unsigned width, precision;
    int number = 256;
    double weight = 242.5;
    printf("Введите ширину поля:\n");
    scanf("%d", &width);
    printf("Значение равно: %*d:\n", width, number);
    printf("Теперь введите ширину и точность:\n");
    scanf("%d %d", &width, &precision);
    printf("Вес = %*.*f\n", width, precision, weight);
    printf("Готово!\n");
    int n;
    printf("Введите три целых числа :\n");
    scanf("%*d %*d %d", &n);
    printf("Последним целым числом было %d\n", n);
    return 0;
}
```

```
// Введите ширину поля:
// 6
// Значение равно:      256:
// Теперь введите ширину и точность:
// 8 3
// Вес =  242.500
// Готово!
// Введите три целых числа :
// 2022 2023 2024
// Последним целым числом было 2024
```

# Адженда

**Строки**

**30 минут**

**Символьный  
ввод-вывод и  
достоверность  
ввода**

**25 минут**

**Массивы.  
Указатели.**

**35 минут**

# getchar() и putchar()

***getchar()*** и ***putchar()*** - функции C, специально предназначенные для символьного ввода-вывода

```
ch = getchar(); == scanf("%c", &ch);  
putchar(ch); == printf("%c", ch);
```

Определены в файле stdio.h.

***Преимущества getchar() и putchar():***

- Скорость
- Компактность
- Простота использования



# Демонстрационная программа

```
#include <stdio.h>
#define SPACE ' ' // кавычка, пробел, кавычка
int main(void)
{
    char ch;
    ch = getchar();    // читать символ
    while (ch != '\n') // пока не встретится конец строки
    {
        if (ch == SPACE) // оставить пробел нетронутым
            putchar(ch); // символ не меняется
        else
            putchar(ch + 1); // изменить другие символы
        ch = getchar();    // получить следующий символ
    }
    putchar(ch); // вывести символ новой строки
    return 0;
}
// CALL ME HAL.
// DBMM HЖ IBM/
```

# Демонстрационная программа

```
#include <stdio.h>
#include <ctype.h> // для доступа к isalpha ()
#define SPACE ' ' // кавычка, пробел, кавычка
int main(void)
{
    char ch;
    while ((ch = getchar()) != '\n')
    {
        if (isalpha(ch)) // если это буква,
            putchar(ch + 1); // отобразить следующую букву
        else // в противном случае
            putchar(ch); // отобразить символ как есть
    }
    putchar(ch); // вывести символ новой строки
    return 0;
}
// Look! It's a programmer!
// Mpp1! Ju't a qsphsbnnfs!
```

# Проверка символьных значений в ctype.h

| Имя функции | Возвращает истинное значение, если аргумент является указанным ниже   |
|-------------|---|
| isalnum()   | Алфавитно-цифровой (буквенный или цифровой)   |
| isalpha()   | Алфавитный  |
| isblank()   | Стандартный пробельный символ (пробел, горизонтальная табуляция либо новая строка) или любой дополнительный символ подобного рода, специфичный для локали                 |
| iscntrl()   | Управляющий символ, такой как <Ctrl+B>  |
| isdigit()   | Цифра   |
| isgraph()   | Любой печатаемый символ, отличный от пробела  |
| islower()   | Символ нижнего регистра   |
| isprint()   | Печатаемый символ   |
| ispunct()   | Символ пунктуации (любой печатаемый символ, отличный от пробела или алфавитно-цифрового символа)  |
| isspace()   | Пробельный символ (символ пробела, новой строки, перевода страницы, возврата каретки, вертикальной или горизонтальной табуляции или, другой символ, определенный локалью) |
| isupper()   | Символ верхнего регистра  |
| isxdigit()  | Символ шестнадцатеричной цифры  |

# Отображение символов в ctype.h

| Имя функции | Действие  |
|-------------|---|
| tolower()   | Если аргумент является символом верхнего регистра, функция возвращает его версию в нижнем регистре; в противном случае она возвращает исходный аргумент |
| toupper()   | Если аргумент является символом нижнего регистра, функция возвращает его версию в верхнем регистре; в противном случае она возвращает исходный аргумент |



# Демо (подсчет слов) – для запуска на ПК скопировать



```
#include <stdio.h>
#include <ctype.h> // для isspace()
#include <stdbool.h> // для bool, true, false
#define STOP '|'
int main(void)
{
    system("chcp 65001");
    char c; // прочитанный символ
    char prev; // предыдущий прочитанный символ
    long n_chars = 0L; // количество символов
    int n_lines = 0; // количество строк
    int n_words = 0; // количество слов
    int p_lines = 0; // количество неполных строк
    bool inword = false; // == true если символ находится внутри слова
    printf("Введите текст для анализа(1 для завершения):\n");
    prev = '\n'; // используется для идентификации полных строк
    while ((c = getchar()) != STOP)
    {
        n_chars++; // считать символы
        if (c == '\n')
            n_lines++;
        if (!isspace(c) && !inword) // считать строки
        {
            inword = true; // начало нового слова
            n_words++; // считать слова
        }
        if (isspace(c) && inword)
            inword = false; // достигнут конец слова
        prev = c; // сохранить значение символа
    }

    if (prev != '\n')
        p_lines = 1;
    printf("символов = %ld, слов = %d, строк = %d, ",
           n_chars, n_words, n_lines);
    printf("неполных строк = %d\n", p_lines);
    return 0;
}

// Reason is a
// powerful servant but
// an inadequate master.
// |
// символов = 55, слов = 9, строк = 3, неполных строк = 0
```

# Буфер

```
#include <stdio.h>
int main(void)
{
    char ch;
    while ((ch = getchar()) != '#')
        putchar(ch);
    return 0;
}
```

## Новый компилятор

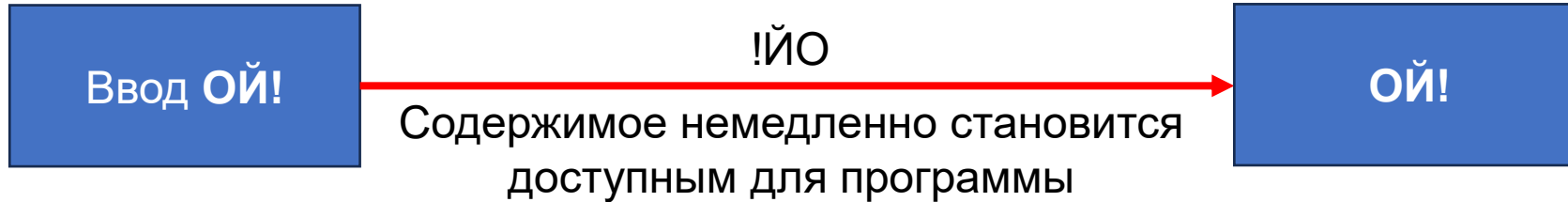
Hello, Mikhail!  
Hello, Mikhail!  
How are # you?  
How are

## Старый компилятор

HHeelloo,, Mmiikkhhaaiill!!

HHooww aarree #

## Небуферизированный ввод



## Буферизированный ввод





# Буфер

## Зачем иметь буферы?

- Передача нескольких символов в виде блока является менее затратной по времени, чем отправка символов по одному.
- В случае опечатки можно воспользоваться средствами коррективки, поддерживаемыми клавиатурой, и исправить опечатку.

С другой стороны, небуферизированный ввод желателен для некоторых интерактивных программ.

Существуют два вида буферизации — полностью буферизированный ввод-вывод и построчно буферизированный ввод-вывод.

1 тип буферизации обычно происходит при файловом вводе. Размер буфера зависит от системы, но наиболее распространены значения 512 и 4096 байтов.

В ANSI C и последующих стандартах C указано, что ввод должен быть буферизированным, но в K&R C выбор изначально возлагался на разработчика компилятора.



# Файлы, потоки и ввод данных с клавиатуры

**Файл** — это область памяти, в которой хранится информация. Концептуально программа на С имеет дело с потоком, а не напрямую с файлом. **Поток** — это идеализированное течение данных, на которое отображается действительный ввод или вывод.

Допустим есть исполняемый файл main.c

**Перенаправление ввода в терминале:**

main.c < in.txt

**Перенаправление вывода в терминале:**

main.c > out.txt

gcc main.c -o main

./main < in.txt > out.txt

```
#include <stdio.h>
int main(void)
{
    char ch;
    while ((ch = getchar()) != '#')
        putchar(ch);
    return 0;
}
```

```
C main.c  out.txt  in.txt
≡ in.txt
1 Hello Mikail!
2 How are # you!

C main.c  out.txt  in.txt
≡ out.txt
1 Hello Mikail!
2 How are
```

# Смешивание числового и СИМВОЛЬНОГО ВВОДА

```
#include <stdio.h>
int main(void)
{
    int ch;          /* выводимый символ */
    int rows, cols; /* количество строк и столбцов */
    printf("Введите символ и два целых числа:\n");
    while ((ch = getchar()) != '\n')
    {
        if (scanf("%d %d", &rows, &cols) != 2)
            break;
        for (int row = 1; row <= rows; row++)
        {
            for (int col = 1; col <= cols; col++)
                putchar(ch);
            putchar('\n'); /* закончить строку и начать новую */
        }
        while (getchar() != '\n')
            continue;
        printf("Введите еще один символ и два целых числа;\n");
        printf("для завершения введите символ новой строки.\n");
    }
    printf("Программа завершена.\n");
    return 0;
}
```

```
// Введите символ и два целых числа:
// с 2 3
// ссс
// ссс
// Введите еще один символ и два целых числа;
// для завершения введите символ новой строки.
// Введите еще один символ и два целых числа;
// для завершения введите символ новой строки.
// с 3 4
// сsss
// сsss
// сsss
// Введите еще один символ и два целых числа;
// для завершения введите символ новой строки.
//
// Программа завершена.
```

# Адженда

**Строки**

**30 минут**

**Символьный  
ввод-вывод и  
достоверность  
ввода**

**25 минут**

**Массивы.  
Указатели.**

**35 минут**

# Массивы

Массив состоит из последовательности элементов одного типа данных.

В объявлении массива указывается, сколько элементов содержит массив и какого типа эти элементы

Квадратные скобки ([]) идентифицируют имена в качестве массивов, а число в квадратных скобках задает количество элементов в массиве.

```
/* несколько объявлений массивов */  
int main(void)  
{  
    float candy[365]; /* массив из 365 элементов типа float */  
    char code[12];    /* массив из 12 элементов типа char */  
    int states[50];   /* массив из 50 элементов типа int */  
}
```

При доступе к элементам в массиве вы указываете отдельный элемент с применением его номера, который также называется индексом. Нумерация элементов начинается с 0.

**Пример:** *candy[0]* — это *первый* элемент массива candy, а *candy [364]* - 365-й, и последний, элемент массива.



# Инициализация массива

```
#include <stdio.h>
#define MONTHS 12
int main(void)
{
    system("chcp 65001");
    int days[MONTHS] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
    int index;
    for (index = 0; index < MONTHS; index++)
        printf("Месяц %d имеет %2d день (дней).\n", index + 1, days[index]);
    return 0;
}
// Месяц 1 имеет 31 день (дней).
// Месяц 2 имеет 28 день (дней).
// Месяц 3 имеет 31 день (дней).
// Месяц 4 имеет 30 день (дней).
// Месяц 5 имеет 31 день (дней).
// Месяц 6 имеет 30 день (дней).
// Месяц 7 имеет 31 день (дней).
// Месяц 8 имеет 31 день (дней).
// Месяц 9 имеет 30 день (дней).
// Месяц 10 имеет 31 день (дней).
// Месяц 11 имеет 30 день (дней).
// Месяц 12 имеет 31 день (дней).
```





# Инициализация массива

А что, если вы забудете инициализировать массив?

```
#include <stdio.h>
#define SIZE 4
int main(void)
{
    int no_data[SIZE]; /* неинициализированный массив */
    int i;
    printf("%2s%14s\n", "i", "no_data[i]");
    for (i = 0; i < SIZE; i++)
        printf("%2d%14d\n", i, no_data[i]);
    return 0;
}

// i      no_data[i]
// 0      1690575848
// 1           533
// 2          100
// 3           0
```



# Инициализация массива

Количество элементов в списке должно соответствовать размеру массива. Но что, если вы подсчитали неправильно?

```
#include <stdio.h>
#define SIZE 4
int main(void)
{
    int some_data[SIZE] = {1492, 1066};
    int i;
    printf("%2s%14s\n", "i", "some_data[i]");
    for (i = 0; i < SIZE; i++)
        printf("%2d%14d\n", i, some_data[i]);
    return 0;
}

// i  some_data[i]
// 0      1492
// 1      1066
// 2         0
// 3         0
```



# Инициализация массива

Когда вы применяете для инициализации массива пустые квадратные скобки, компилятор подсчитывает количество элементов в списке и устанавливает размер массива в полученное число.

```
#include <stdio.h>
int main(void)
{
    const int days[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31};
    int index;
    for (index = 0; index < sizeof days / sizeof days[0]; index++)
        printf("Месяц %2d имеет %d день (дней).\n", index + 1,
               days[index]);
    return 0;
}
// Месяц  1 имеет 31 день (дней).
// Месяц  2 имеет 28 день (дней).
// ...
// Месяц  9 имеет 30 день (дней).
// Месяц 10 имеет 31 день (дней).
```



# Назначенные инициализаторы (C99)

В стандарте C99 добавлена новая возможность — назначенные инициализаторы. Это средство позволяет выбирать, какие элементы будут инициализированы.

```
#include <stdio.h>
#define MONTHS 12
int main(void)
{
    int days[MONTHS] = {31, 28, [4] = 31, 30, 31, [1] = 29};
    int i;
    for (i = 0; i < MONTHS; i++)
        printf("%2d %d\n", i + 1, days[i]);
    return 0;
}
```

```
// 1 31
// 2 29
// 3 0
// 4 0
// 5 31
// 6 30
// 7 31
// 8 0
// 9 0
// 10 0
// 11 0
// 12 0
```



```
int stuff[] = {1, [6] = 23}; // что происходит?  
int staff[] = {1, [6] = 4, 9, 10}; // что происходит?
```

# Присваивание значений элементам массива

```
/* присваивание значений элементам массива */  
#define SIZE 50  
int main(void)  
{  
    int counter, evens[SIZE];  
    for (counter = 0; counter < SIZE; counter++)  
        evens[counter] = 2 * counter;  
    ...  
}
```

```
#define SIZE 5  
int main(void)  
{  
    int oxen[SIZE] = {5, 3, 2, 8}; /* здесь все в порядке */  
    int yaks[SIZE];  
    yaks = oxen; /* не разрешено */  
    yaks[SIZE] = oxen[SIZE]; /* выход за пределы диапазона */  
    yaks[SIZE] = {5, 3, 2, 8}; /* не работает */  
}
```





# Границы массива

```
/* присваивание значений элементам массива */  
#define SIZE 50  
int main(void)  
{  
    int counter, evens[SIZE];  
    for (counter = 0; counter < SIZE; counter++)  
        evens[counter] = 2 * counter;  
    ...  
}
```

```
#define SIZE 5  
int main(void)  
{  
    int oxen[SIZE] = {5, 3, 2, 8}; /* здесь все в порядке */  
    int yaks[SIZE];  
    yaks = oxen; /* не разрешено */  
    yaks[SIZE] = oxen[SIZE]; /* выход за пределы диапазона */  
    yaks[SIZE] = {5, 3, 2, 8}; /* не работает */  
}
```




# Границы массива

```
#include <stdio.h>
#define SIZE 4
int main(void)
{
    int value1 = 44;
    int arr[SIZE];
    int value2 = 88;
    int i;
    printf("value1 = %d, value2 = %d\n", value1, value2);
    for (i = -1; i <= SIZE; i++)
        arr[i] = 2 * i + 1;
    for (i = -1; i < 7; i++)
        printf("%2d %d\n", i, arr[i]);
    printf("value1 = %d, value2 = %d\n", value1, value2);
    printf("адрес arr[-1]: %p\n", &arr[-1]);
    printf("адрес arr[4]: %p\n", &arr[4]);
    printf("адрес value1: %p\n", &value1);
    printf("адрес value2: %p\n", &value2);
    return 0;
}
```

```
// -1 -1
// 0 1
// 1 3
// 2 5
// 3 7
// 4 9
// 5 672
// 6 44
// value1 = 44, value2 = -1
// адрес arr[-1]: 000000d30d7ff8dc
// адрес arr[4]: 000000d30d7ff8f0
// адрес value1: 000000d30d7ff8f8
// адрес value2: 000000d30d7ff8dc
```

# Указание размера массива



```
int n = 5;
int m = 8;
float a1[ 5]; // да
float a2[5*2 + 1]; // да
float a3[sizeof(int) + 1]; // да
float a4[-4]; // нет, размер должен быть > 0
float a5[0]; // нет, размер должен быть > 0
float a6[2.5]; // нет, размер должен быть целым числом
float a7[(int)2.5]; // да, приведение константы float к типу int
float a8[n]; // не было разрешено до появления стандарта C99
float a9[m]; // не было разрешено до появления стандарта C99
```

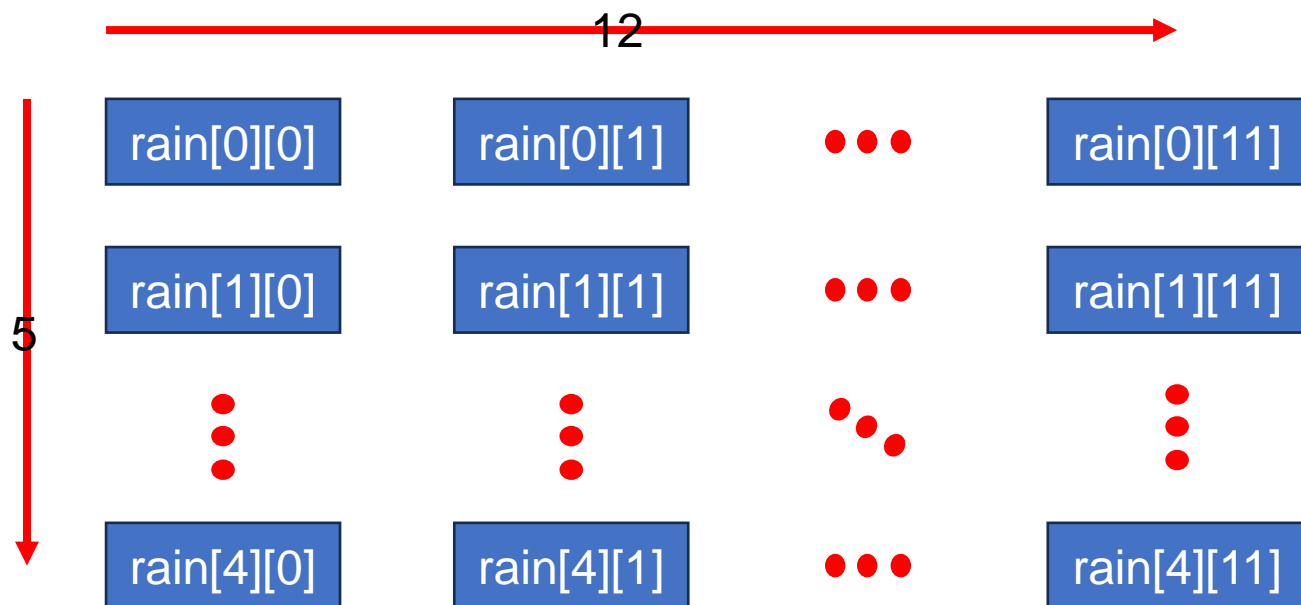
# Многомерные массивы

**Пример:** метеоролог, желает проанализировать данные об осадках за последние пять лет

`float rain[5][12];` // массив из 5 массивов по 12 элементов `float`

`float rain[5][12];` // `rain` - массив, содержащий пять пока невыясненных сущностей

`float rain[5][12];` // массив из 12 значений `float`



# Многомерные массивы (демо ПК)

```
#include <stdio.h>
#define MONTHS 12 // количество месяцев в году
#define YEARS 5   // количество лет, для которых доступны данные
int main(void)
{ // инициализация данными об осадках за период с 2010 по 2014 гг.
    const float rain[YEARS][MONTHS] = {
        {4.3, 4.3, 4.3, 3.0, 2.0, 1.2, 0.2, 0.2, 0.4, 2.4, 3.5, 6.6},
        {8.5, 8.2, 1.2, 1.6, 2.4, 0.0, 5.2, 0.9, 0.3, 0.9, 1.4, 7.3},
        {9.1, 8.5, 6.7, 4.3, 2.1, 0.8, 0.2, 0.2, 1.1, 2.3, 6.1, 8.4},
        {7.2, 9.9, 8.4, 3.3, 1.2, 0.8, 0.4, 0.0, 0.6, 1.7, 4.3, 6.2},
        {7.6, 5.6, 3.8, 2.8, 3.8, 0.2, 0.0, 0.0, 0.0, 1.3, 2.6, 5.2}};

    int year, month;
    float subtot, total;
    printf("ГОД КОЛИЧЕСТВО ОСАДКОВ (в дюймах)\n");
    for (year = 0, total = 0; year < YEARS; year++)
    { // для каждого года суммировать количество осадков за каждый месяц
        for (month = 0, subtot = 0; month < MONTHS; month++)
            subtot += rain[year][month];
        printf("%5d %15.1f\n", 2010 + year, subtot);
        total += subtot; // общая сумма для всех лет
    }
    printf("Среднегодовое количество осадков составляет %.1f дюймов.\n\n", total / YEARS);
    printf("СРЕДНЕМЕСЯЧНОЕ КОЛИЧЕСТВО ОСАДКОВ:\n\n");
    printf(" Янв Фев Мар Апр Май Июн Июл Авг Сен Окт Ноя Дек");
    for (month = 0; month < MONTHS; month++)
    { // для каждого месяца суммировать количество осадков на протяжении годов
        for (year = 0, subtot = 0; year < YEARS; year++)
            subtot += rain[year][month];
        printf("%4.1f ", subtot / YEARS);
    }
    printf("\n");
    return 0;
}
```

```
// ГОД КОЛИЧЕСТВО ОСАДКОВ (в дюймах)
// 2010                32
// 2011                38
// 2012                50
// 2013                44
// 2014                33
// Среднегодовое количество осадков составляет
39.4 дюймов.

// СРЕДНЕМЕСЯЧНОЕ КОЛИЧЕСТВО ОСАДКОВ:

// Янв Фев Мар Апр Май Июн Июл Авг Сен Окт Ноя
Дек
// 7.3 7.3 4.9 3.0 2.3 0.6 1.2 0.3 0.5 1.7 3.6
6.7
```



# Инициализация двумерного массива

|   |   |   |
|---|---|---|
| 5 | 6 | 0 |
|---|---|---|

|   |   |   |
|---|---|---|
| 7 | 8 | 0 |
|---|---|---|

```
int sq[2][3] = {{5,6},{7,8}};
```

|   |   |   |
|---|---|---|
| 5 | 6 | 7 |
|---|---|---|

|   |   |   |
|---|---|---|
| 8 | 0 | 0 |
|---|---|---|

```
int sq[2][3]={5,6,7,8};
```





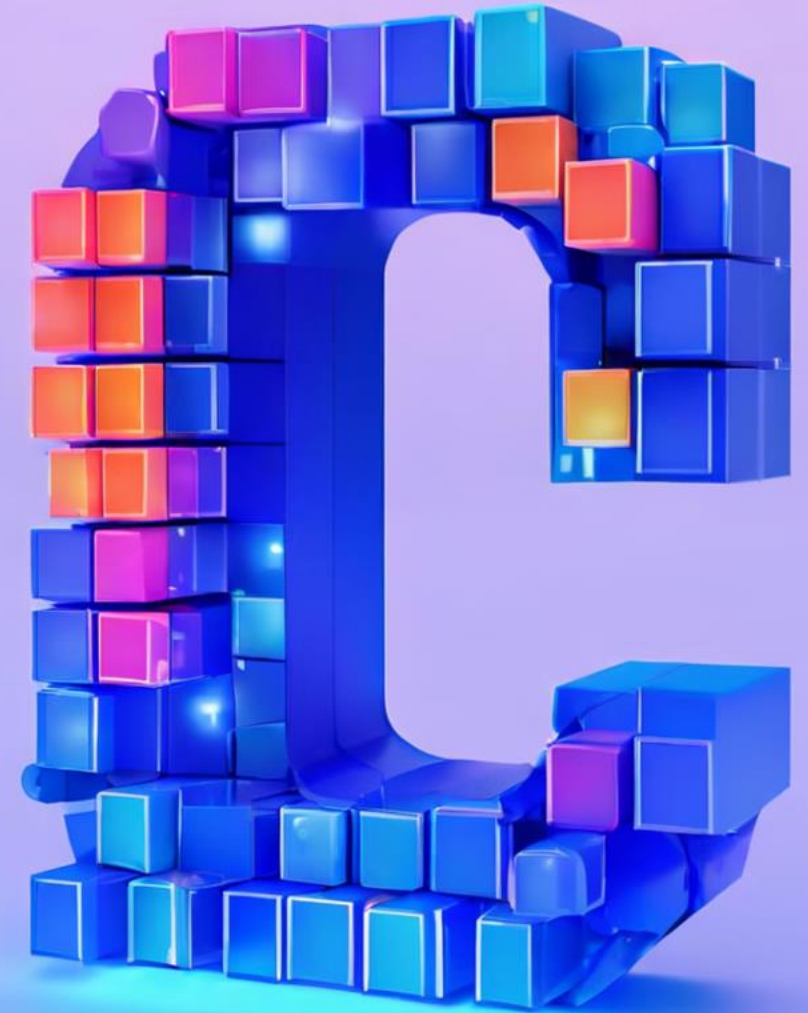
# Большее количество измерений

Все, что было сказано о двумерных массивах, можно распространить на трехмерные массивы и на массивы с большим числом измерений.

Трехмерный массив объявляется следующим образом:

```
int box [10][20][30];
```

Обычно для обработки трехмерного массива применяются три вложенных цикла, для обработки четырехмерного массива-четыре вложенных цикла и т.д.



# Выяснение адресов: операция &

Унарная операция & предоставляет адрес, по которому хранится переменная. Если pooh является именем переменной, то &pooh — адрес этой переменной.

В IBM PC адреса часто задаются в виде шестнадцатеричных значений.

```
#include <stdio.h>
int main(void)
{
    int pooh = 2, bah = 5; /* локальные для main() */
    printf("Внутри main() pooh = %d и &pooh = %p\n", pooh, &pooh);
    printf("Внутри main() bah = %d и &bah = %p\n", bah, &bah);
    mikado(pooh);
    return 0;
}
// Внутри main() pooh = 2 и &pooh = 000000e128bfff80c
// Внутри main() bah = 5 и &bah = 000000e128bfff808
```

# Указатели: знакомство

**Указатель** — это переменная (или в общем случае объект данных), значением которой является адрес в памяти.

```
ptr = &rooh; // присваивает переменной ptr адрес переменной rooh  
ptr = &bah; // переменная ptr указывает на bah вместо rooh
```

Тогда для выяснения значения, хранящегося в переменной bah, можно применить **операцию разыменования** \*

```
val = *ptr; // выяснение значения, на которое указывает ptr
```

Операторы `ptr = &bah;` и `val = *ptr;` вместе эквивалентны следующему оператору:

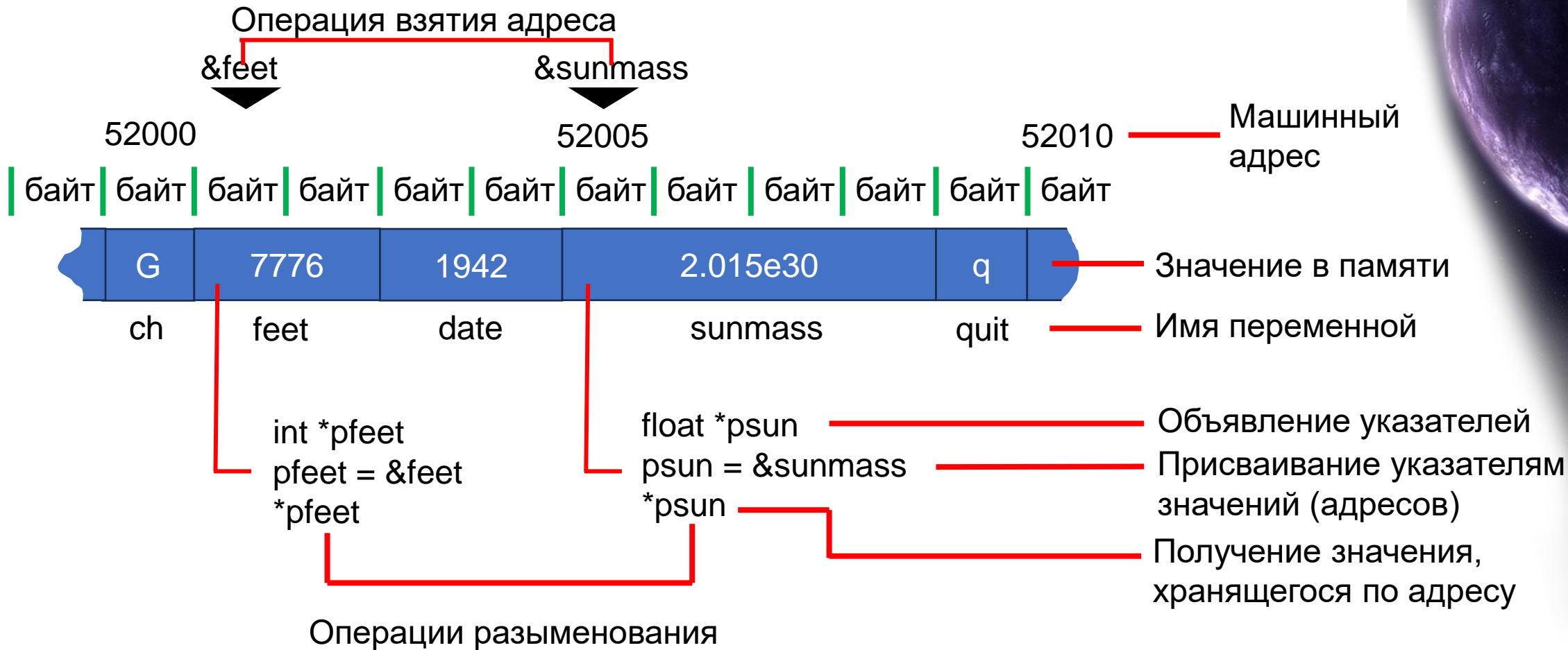
```
val = bah;
```

**Объявление указателей:**

```
int *pi;           // pi - указатель на целочисленную  
переменную  
char *pc;          // pc - указатель на символьную переменную  
float *pf, *pg;    // pf, pg - указатели на переменные с  
плавающей запятой
```



# Объявление и использование указателей



# Указатели и массивы

Система обозначения массивов является просто замаскированным использованием указателей

```
#include <stdio.h>
#define SIZE 4
int main(void)
{
    system("chcp 65001");
    short dates[SIZE];
    short *pti;
    short index;
    double bills[SIZE];
    double *ptf;
    pti = dates; // присваивание указателю адреса массива
    ptf = bills;
    printf("%23s %15s\n", "short", "double");
    for (index = 0; index < SIZE; index++)
        printf("указатели + %d: %10p %10p\n",
            index, pti + index, ptf + index);
    return 0;
}
```

| short          | double                |
|----------------|-----------------------|
| указатели + 0: | fae11ff880 fae11ff860 |
| указатели + 1: | fae11ff882 fae11ff868 |
| указатели + 2: | fae11ff884 fae11ff870 |
| указатели + 3: | fae11ff886 fae11ff878 |



# Указатели и массивы

Система обозначения массивов является просто замаскированным использованием указателей

```
#include <stdio.h>
#define MONTHS 12
int main(void)
{
    int days[MONTHS] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
    int index;
    for (index = 0; index < MONTHS; index++)
        printf("Месяц %2d имеет %d день (дней).\n", index + 1,
               *(days + index)); // то же самое, что и days [index]
    return 0;
}
```

Месяц 1 имеет 31 день (дней).  
Месяц 2 имеет 28 день (дней).  
Месяц 3 имеет 31 день (дней).  
Месяц 4 имеет 30 день (дней).  
Месяц 5 имеет 31 день (дней).  
Месяц 6 имеет 30 день (дней).  
Месяц 7 имеет 31 день (дней).  
Месяц 8 имеет 31 день (дней).  
Месяц 9 имеет 30 день (дней).  
Месяц 10 имеет 31 день (дней).  
Месяц 11 имеет 30 день (дней).  
Месяц 12 имеет 31 день (дней).

Здесь `days` — это адрес первого элемента массива, индекс `days + index` — адрес элемента `days [index]` и `*(days + index)` — значение этого элемента, в точности как `days [index]`. Цикл по очереди ссылается на каждый элемент массива и выводит обнаруженное содержимое.



# Инкрементирование и указатели

```
#include <stdio.h>
int data[2] = {100, 200};
int moredata[2] = {300, 400};
int main(void)
{
    int *p1, *p2, *p3;
    p1 = p2 = data;
    p3 = moredata;
    printf(" *p1 = %d, *p2 = %d, *p3 = %d\n", *p1, *p2, *p3);
    printf("*p1++ = %d, *++p2 = %d, (*p3)++ = %d\n", *p1++, *++p2, (*p3)++);
    printf(" *p1 = %d, *p2 = %d, *p3 = %d\n", *p1, *p2, *p3);
    return 0;
}

//      *p1 = 100,      *p2 = 100,      *p3 = 300
// *p1++ = 100, *++p2 = 200, (*p3)++ = 300
//      *p1 = 200,      *p2 = 200,      *p3 = 301
```

# Операции с указателями

- Присваивание.
- Нахождение значения (разыменование).
- Добавление целого числа к указателю.
- Инкрементирование указателя.
- Вычитание целого числа из указателя.
- Декрементирование указателя.
- Разность.
- Сравнение.



# Операции с указателями (Демо ПК)

```
#include <stdio.h>
int main(void)
{
    int urn[5] = {100, 200, 300, 400, 500};
    int *ptr1, *ptr2, *ptr3;
    ptr1 = urn; // присваивание указателю адреса
    ptr2 = &urn[2]; // то же самое
    // разыменование указателя и получение
    // адреса указателя
    printf("значение указателя, разыменованный указатель, адрес указателя:\n");
    printf("ptr1 = %p, *ptr1 = %d, &ptr1 = %p\n", ptr1, *ptr1, &ptr1);
    // сложение указателей
    ptr3 = ptr1 + 4;
    printf("\nСложение значения int с указателем: \n");
    printf("ptr1 + 4 = %p, * (ptr4 + 3) = %d\n", ptr1 + 4, *(ptr1 + 3));
    ptr1++; // инкрементирование указателя
    printf("\nзначения после выполнения операции ptr1++: \n");
    printf("ptr1 = %p, *ptr1 = %d, &ptr1 = %p\n", ptr1, *ptr1, &ptr1);
    ptr2--; // декрементирование указателя
    printf("\nзначения после выполнения операции --ptr2:\n");
    printf("ptr2 = %p, *ptr2 = %d, &ptr2 = %p\n", ptr2, *ptr2, &ptr2);
    --ptr1; // восстановление исходного значения
    ++ptr2; // восстановление исходного значения
    printf("ptr1 = %p, ptr2 = %p\n", ptr1, ptr2);
    // вычитание одного указателя из другого
    printf("\nвычитание одного указателя из другого:\n");
    printf("ptr2 = %p, ptr1 = %p, ptr2 - ptr1 = %td\n", ptr2, ptr1, ptr2 - ptr1);
    // вычитание целого значения из указателя
    printf("\nвычитание из указателя значения типа int:\n");
    printf("ptr3 = %p, ptr3 - 2 = %p\n", ptr3, ptr3 - 2);
    return 0;
}
```

// значение указателя, разыменованный указатель, адрес указателя:

// ptr1 = 00000061913ff920, \*ptr1 = 100, &ptr1 = 00000061913ff918

// Сложение значения int с указателем:

// ptr1 + 4 = 00000061913ff930, \* (ptr4 + 3) = 400

// значения после выполнения операции ptr1++:

// ptr1 = 00000061913ff924, \*ptr1 = 200, &ptr1 = 00000061913ff918

// значения после выполнения операции --ptr2:

// ptr2 = 00000061913ff924, \*ptr2 = 200, &ptr2 = 00000061913ff910

// ptr1 = 00000061913ff920, ptr2 = 00000061913ff928

// вычитание одного указателя из другого:

// ptr2 = 00000061913ff928, ptr1 = 00000061913ff920, ptr2 - ptr1 = 2

// вычитание из указателя значения типа int:

// ptr3 = 00000061913ff930, ptr3 - 2 = 00000061913ff928

# Разыменование неинициализированного указателя

```
int *pt; // неинициализированный указатель
```

```
*pt = 5; // катастрофическая ошибка
```

**В любом случае, во избежание проблем, никогда не разыменовывайте неинициализированный указатель!**



# Ключевое слово `const`

```
const double PI = 3.14159;
```

Ключевое слово **`const`** дополнительно *позволяет создавать константные массивы, константные указатели и указатели на константы*

```
#define MONTHS 12
```

```
const int days [MONTHS] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
```

```
days[9] = 44; /* ошибка на этапе компиляции */
```

Указатели на константы не могут использоваться для изменения значений.

```
double rates[5] = {88.99, 100.12, 59.45, 183.11, 340.5};
```

```
const double * pd = rates; // pd указывает на начало массива
```

```
*pd = 29.89; // не разрешено
```

```
pd[2] = 222.22; // не разрешено
```

```
rates[0] = 99.99; // разрешено, т.к. rates не является const
```

```
pd++; /* теперь pd указывает на rates[1] -- разрешено */
```



# Ключевое слово `const`

Существует несколько правил, которые вы должны соблюдать, присваивая указатели и используя ключевое слово `const`. Прежде всего, указателю на константу допускается присваивание адреса либо константных, либо не константных данных:

```
double rates [5] = {88.99, 100.12, 59.45, 183.11, 340.5};  
const double locked[4] = {0.08, 0.075, 0.0725, 0.07};  
const double *pc = rates;    // допустимо  
pc = locked;                 // допустимо  
pc = &rates[3];              // допустимо
```

Тем не менее, обычным указателям могут быть присвоены только адреса неконстантных данных:

```
double rates[5] = {88.99, 100.12, 59.45, 183.11, 340.5};  
const double locked[4] = {0.08, 0.075, 0.0725, 0.07};  
double * pnc = rates;        // допустимо  
pnc = locked;                // не допустимо  
pnc = &rates[3];             // допустимо
```





# Ключевое слово `const`

Существуют и другие варианты использования `const`. К примеру, вы можете объявить и инициализировать указатель таким образом, чтобы его нельзя было заставить указывать на что-нибудь другое. Хитрость в том, где размещено ключевое слово `const`:

```
double rates[5] = {88.99, 100.12, 59.45, 183.11, 340.5};  
double * const pc = rates; // pc указывает на начало массива  
pc = &rates[2];           // не разрешено указывать на что-нибудь другое  
*pc = 92.99;              // все в порядке -- изменяется rates[0]
```

`const` можно использовать дважды, чтобы создать указатель, который не допускает изменения ни адреса, куда он указывает, ни указываемого с помощью него значения:

```
double rates[5] = {88.99, 100.12, 59.45, 183.11, 340.5};  
const double * const pc = rates;  
pc = &rates[2];           // не разрешено  
pc = &rates [92.99];      // не разрешено
```



# Указатели и многомерные массивы

```
int zipro[4][2]; /* массив из массивов типа int */
```

- `zipro` — адрес первого элемента массива, `zipro` имеет тоже значение, что и `&zipro [0]`.
- `zipro [0]` является массивом из двух целых чисел, следовательно, `zipro [0]` имеет тоже значение, что и `&zipro [0][0]`, т.е. адрес его первого элемента - значения `int`.
- `zipro + 1` имеет значение, не совпадающее с `zipro [0] + 1`.
- `*(zipro[0])` представляет значение, хранящееся в `zipro[0][0]`.
- `*zipro` представляет значение своего первого элемента (`zipro[0]`), но `zipro[0]` сам по себе — адрес значения `int`. Это адрес `&zipro[0][0]`, так что `*zipro` является `&zipro[0][0]`.
- Применение операции разыменования к обоим выражениям предполагает, что `**zipro` равно `*&zipro[0][0]`, что сокращается до `zipro[0][0]`.



# Указатели и многомерные массивы

```
#include <stdio.h>
int main(void)
{
    int zippo[4][2] = {{2, 4}, {6, 8}, {1, 3}, {5, 7}};
    printf(" zippo = %p, zippo + 1 = %p\n", zippo, zippo + 1);
    printf("zippo[0] = %p, zippo[0] + 1 = %p\n", zippo[0], zippo[0] + 1);
    printf(" *zippo = %p, *zippo + 1 = %p\n", *zippo, *zippo + 1);
    printf("zippo[0][0] = %d\n", zippo[0][0]);
    printf(" *zippo[0] = %d\n", *zippo[0]);
    printf(" **zippo = %d\n", **zippo);
    printf(" zippo[2][1] = %d\n", zippo[2][1]);
    printf("*(*(zippo+2)+1) = %d\n", *(*(zippo + 2) + 1));
    return 0;
}

//      zippo = 000000d2975ff990,      zippo + 1 = 000000d2975ff998
//  zippo[0] = 000000d2975ff990, zippo[0] + 1 = 000000d2975ff994
//      *zippo = 000000d2975ff990,      *zippo + 1 = 000000d2975ff994
//          zippo[0][0] = 2
//          *zippo[0] = 2
//          **zippo = 2
//          zippo[2][1] = 3
//      *(*(zippo+2)+1) = 3
```

# Указатели на многомерные массивы

```
int main(void)
{
    int zippo[4][2] = {{2, 4}, {6, 8}, {1, 3}, {5, 7}};
    int(*pz)[2];
    pz = zippo;
    printf(" pz = %p, pz + 1 = %p\n", pz, pz + 1);
    printf("pz[0] = %p, pz[0] + 1 = %p\n", pz[0], pz[0] + 1);
    printf(" *pz = %p, *pz + 1 = %p\n", *pz, *pz + 1);
    printf("pz[0][0] = %d\n", pz[0][0]);
    printf("*pz[0] = %d\n", *pz[0]);
    printf("**pz = %d\n", **pz);
    printf(" pz[2][1] = %d\n", pz[2][1]);
    printf("*(*(pz+2) +1) = %d\n", (*(pz + 2) + 1));
    return 0;
}

//      pz = 000000ab4ddffdc0,      pz + 1 = 000000ab4ddffdc8
// pz[0] = 000000ab4ddffdc0, pz[0] + 1 = 000000ab4ddffdc4
//  *pz = 000000ab4ddffdc0,   *pz + 1 = 000000ab4ddffdc4
//      pz[0][0] = 2
//          *pz[0] = 2
//              **pz = 2
//      pz[2][1] = 3
//  *(*(pz+2) +1) = 3
```

# Совместимость указателей

Правила присваивания одного указателя другому строже таких правил для числовых типов.

```
int n = 5;
```

```
double x;
```

```
int *pi = &n;
```

```
double *pd = &x;
```

```
x = n;           // неявное преобразование типа
```

```
pd = pi;         // ошибка на этапе компиляции
```





# Совместимость указателей

Предположим, что есть следующие объявления:

```
int * pt;  
int (*pa) [3];  
int ar1[2][3];  
int ar2[3][2];  
int **p2;                // указатель на указатель
```

Взгляните на показанный далее код:

```
pt = &ar1[0][0];           // оба - указатели на int  
pt = &ar1[0];              // оба - указатели на int  
pt = ar1;                  // недопустимо  
pa = ar1;                  // оба - указатели на int [3]  
pa = ar2;                  // недопустимо  
pt = &pt;                  // оба - указатели на int  
*p2 = ar2[0];              // оба - указатели на int  
p2 = ar2;                  // недопустимо
```





# Совместимость указателей

В целом, многократные операции разыменования сложны. Например, рассмотрим следующий фрагмент кода:

```
int x = 20;  
const int y = 23;  
int *p1 = &x;  
const int *p2 = &y;  
const int **pp2;  
p1 = p2; // небезопасно — присваивание константного значения неконстантному  
p2 = p1; // допустимо -- присваивание неконстантного значения константному  
pp2 = &p1; // небезопасно -- присваивание вложенных типов указателей
```

```
const int **pp2;  
int *p1;  
const int n = 13;  
pp2 = &p1; // разрешено, но квалификатор const игнорируется  
*pp2 = &n; // допустимо, оба const, но p1 устанавливается указывающим на n  
*p1 = 10; // допустимо, но производится попытка изменить константу n
```

