

---

# Указатели, функция `scanf`, передача массивов в функцию

А. Г. Fenster, `fenster@fenster.name`

26 февраля 2010 г.

Конспект семинара №3 по программированию для студентов 1 курса ММФ НГУ. Помните, что чтение конспекта не делает посещение соответствующего семинара необязательным! О найденных ошибках и неточностях, пожалуйста, пишите мне по адресу `fenster@fenster.name`. Также буду рад получить любые комментарии по поводу этого текста.

## 1 Повторение

**Задача.** Выполнить циклический сдвиг элементов одномерного массива влево:  $1\ 2\ 3\ 4\ 5 \longrightarrow 2\ 3\ 4\ 5\ 1$

**Задача.** Выполнить циклический сдвиг элементов одномерного массива вправо:  $1\ 2\ 3\ 4\ 5 \longrightarrow 5\ 1\ 2\ 3\ 4$

**Задача.** Дана матрица  $N \times N$  (в двумерном массиве) и вектор длины  $N$  (в одномерном). Умножьте матрицу на вектор (или вектор на матрицу — зависит от того, считать вектор столбцом или строкой).

## 2 Указатели

Любой (за редкими исключениями, которые мы сейчас не рассматриваем) переменной соответствует место в памяти в которой она размещается. Определить, где размещена переменная, можно при помощи *оператора взятия адреса* `&`, который может быть применён к переменной. По-

лученный адрес — некое значение, однозначно (для данной программы) определяющее место в памяти, в котором хранится данная переменная.

Любое значение в языке C имеет некоторый тип (`int`, `float`, ...); должны иметь тип и адреса ячеек памяти. Хотя физически они представляют собой целые числа, в языке выделены специальные типы для хранения адресов: адрес переменной типа `T` — значение типа `T *`. Итак, можно завести переменную типа, к примеру, `int *` и присвоить ей адрес некоторого значения типа `int` в памяти — т. е. адрес места в памяти, где это значение размещено. Такая переменная называется *указателем* на `int`.

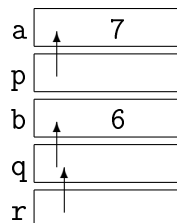
Если в некоторой вымышленной операционной системе байты доступной программе памяти нумеруются, начиная с единицы, то при выполнении следующего кода возможна ситуация, показанная на рисунке. Одна клетка соответствует четырём байтам — обычному размеру типа `int` и размеру указателя на машинах архитектуры x86.

```
int a = 7;    /* переменная a размещена в ячейке 1, содержит число 7 */
int *p = &a; /* переменная p размещена в ячейке 5, содержит адрес переменной a (1)*/
int b = 6;    /* переменная b размещена в ячейке 9, содержит число 6 */
int *q = &b;  /* переменная q размещена в ячейке 13, содержит адрес переменной b (9) */
int **r = &q; /* переменная r размещена в ячейке 17, содержит адрес переменной q (13) */
```

a	1	2	3	4
			7	
p	5	6	7	8
			1	
b	9	10	11	12
			6	
q	13	14	15	16
			9	
r	17	18	19	20
			13	
	21	22	23	24

Под адресом переменной мы понимаем адрес первой ячейки, которую занимает переменная: в примере выше `a` занимает четыре байта (номера 1, 2, 3, 4), но адресом её считается 1.

Это был первый и последний раз, когда мы придумывали «номера ячеек памяти», в будущем подобные картинки мы будем рисовать, используя стрелки. Сравните нарисованное выше и ниже:



*Операция разыменования* — получение значения ячейки памяти по её адресу, на рисунке разыменованию соответствует один проход по стрелке. Операция разыменования обозначается символом `*` слева от разыменуемого адреса. В примере выше выражение `*p` имеет значение 7, выражение `*q` — 6, а выражение `*r` равно адресу переменной `b` (таким образом, `**r` имеет значение 6).

При объявлении нескольких переменных `int *x, y, *z`; «звёздочки» всегда относятся к именам переменных, а не к типу (таким образом, `y` будет иметь тип `int`, а `z` — тип `int *`).

Результат операции разыменования является *левосторонним значением* (lvalue), т. е. может стоять в левой части оператора присваивания. В примере на предыдущем рисунке присваивания

```
b = 8;
*q = 8;
**r = 8;
```

выполняют одно и то же действие: изменяют значение переменной `b`.

### 3 Вызов функции

В языке C вызов функции происходит *по значению*. Это означает, что если определена функция

```
int f(int a)
{
    return (a + 1);
}
```

и она вызывается из программы так:

```
x = f(7);
```

то значение 7 будет скопировано в область памяти, соответствующую переменной `a`, а затем результат функции будет скопирован в переменную `x`. Параметр `a` является локальной переменной для функции `f`, начальное значение которого определяется вызовом функции; функция вольна делать с ним что угодно:

```
void g(int a)
{
    a++;
}

...
g(6);

...
int x = 5;
g(x);
/* здесь x == 5 */
```

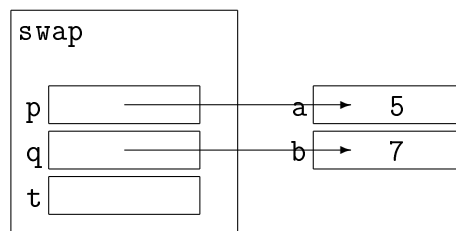
Изменяя переменную `a`, функция меняет лишь свою копию данного ей значения, что никак не может отразиться ни на переменной `x` во втором вызове, ни тем более на константе `6`.

## 4 Передача адресов значений в функцию

Первое (и, возможно, наиболее частое) применение указателей в С — создание функций, которые могут изменять значения переданных им фактически параметров. Смысл трюка заключается в том, что функции при вызове передаётся **адрес** интересующего значения, который сам по себе не изменяется, но при этом функция может изменить находящееся по этому адресу значение. Классический пример — функция `swap`, меняющая местами значения двух переменных:

```
void swap(int *p, int *q) /* принимает адреса переменных */
{
    int t = *p; /* меняет значения по этим адресам */
    *p = *q;
    *q = t;
}

...
int a = 5, b = 7;
swap(&a, &b); /* передаём адреса интересующих переменных */
```



## 5 Функция `scanf`

Аналогичным образом вынуждена действовать и функция `scanf`, которая читает с клавиатуры (точнее, из потока стандартного ввода) значения указанных типов и помещает их по адресам, переданным ей в качестве параметров. Пример:

```
int a;
scanf("%d", &a); /* передаём адрес переменной! */
```

Функция `scanf` в нормальном случае возвращает количество успешно прочитанных значений. Позже мы будем активно это использовать (например, при чтении из файла, пока не достигнут его конец).

## 6 Указатели и массивы

Оказывается, имя любого массива — это по сути неизменяемый указатель на начало (нулевой элемент) этого массива. Более того, добавление числа к указателю даёт новый указатель, сдвинутый на такое же количество элементов этого типа вправо. Так как массивы всегда занимают в памяти непрерывный блок ячеек, получаем, что для массива `int A[5]` адрес  $i$ -го элемента `&A[i]` совпадает с `A + i`. Вообще говоря, выполняется тождество

```
A[i] == *(A + i)
```

Более того, благодаря коммутативности сложения допускается (но не приветствуется) даже запись `i[A]` (и даже `0[A]`, `1[A]` и так далее).

## 7 Передача массивов в функцию

При передаче массивов в функцию два следующих описания полностью эквивалентны:

<pre>void f(int A[5]) {     ... }</pre>		<pre>void f(int *A) {     ... }</pre>
---	--	---------------------------------------

Более того, число 5, указанное в первом случае, никакого значения не имеет и функции не доступно. Проще говоря, при передаче в функцию

массива функция получает лишь адрес его начала, а определить количество элементов в нём не может никак — это забота программиста: длину обычно передают отдельно.

Например, следующая функция суммирует указанное количество элементов массива:

```
int sum(int *p, int n)
{
    int i = 0;
    int result = 0;
    for (i = 0; i < n; i++)
    {
        result += p[i]; // то же, что *(p + i), т. е. i-й элемент массива
    }
    return result;
}
```

Здесь мы обращаемся с `p` как с массивом. Возможен и другой вариант: с `p` можно работать как с указателем.

```
int sum(int *p, int n)
{
    int result = 0;
    while (n > 0)
    {
        result += *p;
        p++;
        n--;
    }
    return result;
}
```

Здесь `*p` получает значение элемента массива, на который указывает `p`, а `p++` сдвигает `p` вправо на один элемент массива.

## 8 Проверочное задание («пятиминутка»)

Тема: передача массивов в функцию. Какие из перечисленных вызовов функции `sum` вернут определённый результат? (см. [задание](#))