

# Семестр 1. Лекция 2. Указатели.

Евгений Линский

15 Сентября 2017

| Название<br>типа   | Кол-во байт<br>для хранения | Диапазон   |
|--------------------|-----------------------------|--|
| char               | 1                           | $-2^7 \dots 2^7 - 1$                             |
| short              | 2                           | $-2^{15} \dots 2^{15} - 1$                       |
| int                | 4                           | $-2^{31} \dots 2^{31} - 1$                       |
| long               | 4                           | $-2^{31} \dots 2^{31} - 1$                       |
| long long          | 8                           | $-2^{63} \dots 2^{63} - 1$                       |
| unsigned char      | 1                           | $0 \dots 2^8 - 1$                                |
| unsigned short     | 2                           | $0 \dots 2^{16} - 1$                             |
| unsigned int       | 4                           | $0 \dots 2^{32} - 1$                             |
| unsigned long      | 4                           | $0 \dots 2^{32} - 1$                             |
| unsigned long long | 8                           | $0 \dots 2^{64} - 1$                             |
| float              | 4                           | $1,4 \cdot 10^{-45} \dots 3,4 \cdot 10^{38}$     |
| double             | 8                           | $4,94 \cdot 10^{-324} \dots 1,79 \cdot 10^{308}$ |

В чем подвох?

- 1 На самом деле размеры типов зависят от платформы (процессор, ОС, компилятор)
- 2 `int` — “естественный” тип (компьютеру проще работать: ширина регистров, особенности набора инструкций)
- 3 На самом деле, например:  
$$\text{sizeof}(\text{short}) \leq \text{sizeof}(\text{int}) \leq \text{sizeof}(\text{long})$$
- 4 `sizeof` — оператор языка (не функция), во время компиляции заменяется на размер типа

- ▶ Представление отрицательных целых чисел (дополнительный код)
- ▶ Представление чисел с плавающей точкой (floating point)
- ▶ Приведение типов (явное, неявное)

NB! Будет тест.

Одномерные:

```
int array[10]; // размер 10*sizeof(int)
//Инициализация:
int array[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
int array[10] = {0}; // обнулить
int array[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
//для типа char:
char array[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
char array[] = {'H', 'e', 'l', 'l', 'o'};
char array[] = "Hello"; // размер?
```

Двумерные:

```
int M[10][10];
int M[2][2] = { {1,2} , {3,4} };
```

```
int array[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};  
int a = array[index];  
a[10] = 1; a[-1] = 1;
```

## “Труднонаходимые” ошибки

- 1 Выход за пределы массива не контролируется компилятором
- 2 Исходы:
  - программа корректно отработала (не задела данные)
  - программа некорректно (зависла, неверный ответ) отработала (задела данные)
  - ОС аварийно завершила программу (задела чужие данные)
- 3 Исход зависит от того, что (данные, другие программы) в памяти в данный момент

- 1 Указатель (pointer) — число, адрес (т.е. смещение от начала) соответствующего элемента в памяти
- 2 `int* p;` — указатель на ячейку, в которой хранится `int` (в 'p' будет храниться адрес)
- 3 Количество байт для хранения указателя зависит от архитектуры компьютера (на x86 сейчас — 64 бита)
- 4 `sizeof(int*) == sizeof(char*) == sizeof(double*)` etc

```
int a = 42;  
int *p = &a; // & - взять адрес a  
int b = *p; // взять значение по адресу p (разыменовать)  
printf("%p", p); // вывести адрес
```



Сдвиг зависит от типа объекта, на который указывает указатель.

```
int array[5] = {1, 2, 3, 4, 5};  
char str[] = "hello";  
int *pi = array; // pi = &array[0]  
char *pc = str; // pc = &str[0]  
pi += 1; // сдвиг адреса на sizeof(int)  
pc += 1; // сдвиг адреса на sizeof(char)
```

array[i] == i[array]:

```
array[i] --> *(array + i)  
i[array] --> *(i + array)
```

## Различие между разными видами указателей

```
char str[4];  
char *pc = &c[0]; //&c[0] - адрес первого элемента массива,  
или адрес массива  
int *pi = pc; // C - ok, C++ - error (разные типы)  
int *pi = (int*)p; // C - ok, C++ - ok
```

```
char array[10];  
char *p;  
p = array; // в p передается адрес массива (адрес нулевого  
элемента)  
array = p; // это ошибка
```

```
void swap(double a, double b){
    double tmp = a;
    a = b;
    b = tmp;
}
int main() {
    double c = 3; double d = 4;
    swap(c, d);
    return 0;
}
```

```
void swap(double a, double b){
    double tmp = a;
    a = b;
    b = tmp;
}
int main() {
    double c = 3; double d = 4;
    swap(c, d);
    return 0;
}
```

- ❗ Ничего не получится.

```
void swap(double a, double b){
    double tmp = a;
    a = b;
    b = tmp;
}
int main() {
    double c = 3; double d = 4;
    swap(c, d);
    return 0;
}
```

1. Ничего не получится.
2. Функция работает с копиями параметров (a и b поменяются, c и d нет).

```
void swap(double *pa, double *pb){
    double tmp = *pa;
    *pa = *pb;
    *pb = tmp;
}
int main() {
    double c = 3; double d = 4;
    swap(&c, &d);
    return 0;
}
```

Передать в функцию большой объект и не копировать его!

```
char str[] = "Hello";  
int l = strlen(str);
```

```
int strlen(char* ptr){
    int len = 0;
    while (ptr[len] != '\0'){
        ++len;
    }
    return len;
}
```

```
int strlen(char* ptr){
    char* p = ptr;
    while (*p != '\0'){
        ++p;
    }
    return p - ptr;
}
```



```
int strlen(char* ptr){
    int len = 0;
    while (ptr[len] != '\0'){
        ++len;
    }
    return len;
}
```

```
int strlen(char* ptr){
    char* p = ptr;
    while (*p != '\0'){
        ++p;
    }
    return p - ptr;
}
```

- 1 ptr[len] -> \*(ptr+len) одно сложение!

```
int strlen(char* ptr){
    int len = 0;
    while (ptr[len] != '\0'){
        ++len;
    }
    return len;
}
```

```
int strlen(char* ptr){
    char* p = ptr;
    while (*p != '\0'){
        ++p;
    }
    return p - ptr;
}
```

- 1 ptr[len] -> \*(ptr+len) одно сложение!
- 2 '\0' — символ с кодом 0.

```
int strlen(char* ptr){
    int len = 0;
    while (ptr[len] != '\0'){
        ++len;
    }
    return len;
}
```

```
int strlen(char* ptr){
    char* p = ptr;
    while (*p != '\0'){
        ++p;
    }
    return p - ptr;
}
```

- 1 ptr[len] -> \*(ptr+len) одно сложение!
- 2 '\0' — символ с кодом 0.
- 3 (p - ptr) — длина строки (складывать указатели нельзя) .