

## Лекция 2. Указатели.

Евгений Линский

Название типа	Кол-во байт 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?

# Зависимость от платформы

- ❶ На самом деле размеры типов зависят от платформы (процессор, ОС, компилятор)
- ❷ `int` — “естественный” тип (компьютеру проще работать: ширина регистров, особенности набора инструкций)
- ❸ На самом деле, например:  
 $\text{sizeof}(\text{short}) \leq \text{sizeof}(\text{int}) \leq \text{sizeof}(\text{long})$
- ❹ `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:

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

Двумерные:

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

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

“Труднонаходимые” ошибки

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

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

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



Сдвиг зависит от типа объекта, на который указывает указатель.

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

array[i] == i[array]:

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

# Различие между разными видами указателей

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

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

# Применение - 1

```
1  void swap(double a, double b){
2      double tmp = a;
3      a = b;
4      b = tmp;
5  }
6  int main() {
7      double c = 3; double d = 4;
8      swap(c, d);
9      return 0;
10 }
```

# Применение - 1

```
1  void swap(double a, double b){
2      double tmp = a;
3      a = b;
4      b = tmp;
5  }
6  int main() {
7      double c = 3; double d = 4;
8      swap(c, d);
9      return 0;
10 }
```

❶ Ничего не получится.

# Применение - 1

```
1 void swap(double a, double b){
2     double tmp = a;
3     a = b;
4     b = tmp;
5 }
6 int main() {
7     double c = 3; double d = 4;
8     swap(c, d);
9     return 0;
10 }
```

- ❶ Ничего не получится.
- ❷ Функция работает с копиями параметров (а и b поменяются, с и d нет).

# Применение - 1

```
1  void swap(double *pa, double *pb){  
2      double tmp = *pa;  
3      *pa = *pb;  
4      *pb = tmp;  
5  }  
6  int main() {  
7      double c = 3; double d = 4;  
8      swap(&c, &d);  
9      return 0;  
10 }
```

Передать в функцию большой объект и не копировать его!

```
1 char str[] = "Hello";  
2 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;  
}
```

- ❶ 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;  
}
```

- ❶ ptr[len] -> \*(ptr+len) одно сложение!
- ❷ '\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) .