

---

# Язык Си. Часть 5

---

ПРИВЕДЕНИЕ ТИПОВ. СИМВОЛЫ И СТРОКИ.



---

# Приведение ТИПОВ

---

# Приведение типов

---

Некоторые операции в зависимости от своих операндов могут инициировать преобразование значений из одного типа в другой.

Приведение типов (cast) в языке Си может быть **явным** (explicit) и **неявным** (implicit).

При **явном** приведении типов перед выражением ставится имя нужного типа в круглых скобках.

Пример: `float k = (float) 1/3; // k = 0.333...`

Неявное приведение типов выполняется компилятором автоматически по указанным далее правилам.

# Преобразования целых типов

---

Знаковые и беззнаковые значения типа **char**, **short int** и **enum** могут использоваться в выражениях везде, где разрешено применение целых чисел. Если тип `int` позволяет представить все значения исходного типа операнда, то операнд приводится к **int**, в противном случае— к **unsigned int**. Эта процедура называется **расширением целочисленного типа** (integral promotion).

При преобразовании `signed`-типа  $A$  к `unsigned`-типу  $B$  если длина  $s$  битов  $A > B$ , то лишние левые биты отбрасываются, иначе – недостающие заполняются нулями.

В результате приведения `unsigned`-типа к `signed` его значение не меняется, если оно представимо в этом новом типе; в противном случае результат зависит от реализации.

# Преобр-е «целое – веществ-е»

---

**При преобразовании из вещественного типа в целочисленный дробная часть числа отбрасывается;** если полученное при этом значение нельзя представить величиной заданного целочисленного типа, то результат не определен.

Если число преобразуется из **целого в вещественное** и находится в допустимом диапазоне, но представляется в новом типе недостаточно точно, то результатом будет большее или меньшее ближайшее значение нового типа. Если результат выходит за границы диапазона допустимых значений, результат не определен.

# Преобр-е вещественных типов

---

При преобразовании из вещественного типа меньшей точности к типу большей точности число не изменяется.  
(float -> double)

Если преобразование выполняется от большей точности к меньшей и число остается в допустимых пределах нового типа, то результатом будет большее или меньшее ближайшее значение нового типа. Если результат выходит за границы допустимого диапазона, результат не определен:

```
double R = 2e+100;
```

```
float R2 = R; /* Исключение типа Floating overflow –  
переполнение типа float */
```

х	у	Результат деления	Пример
делимое	делитель	частное	$x = 15 \ y = 2$
int	int	int	$15/2=7$
int	float	float	$15/2=7.5$
float	int	float	$15/2=7.5$

# Арифметические преобразования

Во-первых, **если один операнд имеет тип:** **то другой преобраз-ся в:**

	long double	long double
Иначе,	double	double
Иначе,	float	float
Иначе,	Для обоих операндов выполняется расширение целого типа (все приводится к <b>unsigned/signed int</b> )	
Иначе,	unsigned long int	unsigned long int
Иначе,	<b>long int</b> , а другой — <b>unsigned int</b> , то если long int представляет все значения unsigned int, то unsigned int приводится к long int; иначе оба операнда преобразуются в unsigned long int.	
Иначе,	long int	long int
Иначе,	unsigned int	unsigned int



# Преобразования указателей

---

Любой указатель можно привести к типу **void \*** без потери информации. Если результат подвергнуть обратному преобразованию, получится исходный указатель. Указатели типа `void *` можно употреблять совместно с указателями любого типа в операциях присваивания и сравнения каким угодно образом.

При преобразовании указателей других типов по факту ничего не происходит, но транслятор «запоминает», что операции адресной арифметики и разыменования нужно выполнять в соответствии с новым типом данных.

---

# Работа со строками и символами

---

# Кодировки СИМВОЛОВ

---

Для представления текста в цифровом формате необходимо придумать систему кодирования, в которой каждой букве соответствовал бы уникальный двоичный код. Свои коды понадобятся и для цифр, и для знаков препинания, поскольку без них в тексте не обойтись.

**Кодирование** – это представление чего-либо чем-нибудь другим.

**Кодировка (набор символов, чарсет, charset, кодовая страница)** – это набор правил, описывающий способ перевода одного представления в другое, т.е. таблица кодов, закрепляющая за символом набор битов.

# Кодировка ASCII 1967 год

---

– американский стандартный код для обмена информацией (**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange).

ASCII — это **7-битовая кодировка**. Ее коды принимают значения от 0000000 до 1111111 или в шестнадцатеричном выражении от 00h до 7Fh.

Первые **33 кода** – управляющие и непечатные символы (например: `'\t' – 09h`, `'\n' – 0Ah`), остальные – буквы латинского алфавита, цифры и знаки препинания.

Для кодирования национальных символов используют расширенную ASCII-таблицу и коды 128-255. Для Windows это кодировка Windows **CP-1251**. Кодировка букв в консоли Windows по умолчанию **MS-DOS CP-866**.

Кириллицу также содержат кодовые страницы ISO-8859-5 (Linux), CP KOI-8U и CP KOI-8R (СССР). Для других языков существуют свои кодовые таблицы.

Код	Символ	Код	Символ	Код	Символ	Код	Символ
32	пробел	56	8	80	P	104	H
33	!	57	9	81	Q	105	I
34	"	58	:	82	R	106	J
35	#	59	;	83	S	107	K
36	\$	60	<	84	T	108	L
37	%	61	=	85	U	109	m
38	&	62	>	86	V	110	n
39	'	63	?	87	W	111	o
40	(	64	@	88	X	112	p
41	)	65	A	89	Y	113	q
42	*	66	B	90	Z	114	r
43	+	67	C	91	[	115	s
44	,	68	D	92	\	116	t
45	-	69	E	93	]	117	u
46	.	70	F	94	^	118	v
47	/	71	G	95	_	119	w
48	0	72	H	96	`	120	x
49	1	73	I	97	A	121	y
50	2	74	J	98	b	122	z
51	3	75	K	99	c	123	{
52	4	76	L	100	d	124	
53	5	77	M	101	e	125	}
54	6	78	N	102	f	126	~
55	7	79	O	103	g	127	del

Полная ASCII-таблица есть в файле [ascii.pdf](#). Там же есть и расширенные таблицы 866 и CP-1251.




# Unicode

1988 год

---

– таблица кодирования символов, которая содержит **1 114 112** кодов для символов всех языков.

Первые 128 символов — с кодами от 0000h до 007Fh — совпадают с символами ASCII.

Unicode в первую и главную очередь определяет таблицу пунктов для символов. Это такой способ сказать «65 – А, 66 – В, 9731 – ». Как эти пункты кодируются в байты **зависит от конкретной кодировки**. Для представления 1 114 112 значений двух байт недостаточно. Трех достаточно, но 3 – странное число, так что 4 является комфортным минимумом.

**UTF-32** – это кодировка, которая переводит все символы в наборы из 32 бит.

**UTF-16 и UTF-8** являются кодировками с переменной длиной кодирования. Если символ может быть закодирован одним байтом, UTF-8 закодирует его одним байтом. Если нужно 2 байта, то используется 2 байта. Кодировка сообщает старшими битами, сколькими битами кодируется текущий символ.

**UTF-16** является компромиссом: все символы как минимум двухбайтные, но их размер может увеличиваться до 4 байт, если нужно.

# Тип char

---

Переменная типа char занимает 1 байт памяти и хранит целое число, обозначающее код символа в какой-либо стандартной кодировке.

Отсюда следует, что, во-первых, с типом char можно работать, как с обычным целым числом. А во-вторых, что значения типа **int** тоже можно воспринимать как коды символов.

В случае с типом int по умолчанию подразумевается модификатор знака signed. Но что касается типа char, то в стандарте указано, что **тип char должен совпадать либо с signed char, либо с unsigned char, и это зависит от компилятора!** То есть в общем случае, типы **signed char, unsigned char и char – это 3 разных типа!\***

# Функции для работы с char

## Ввод/вывод:

---

1. Указание типа в printf/scanf - **%c**. Пример: вывод всей кодовой таблицы:

```
int main() {  
    int i;  
    for( i = 0; i < 256; i++)  
        printf("symbol: %c, code = %d\n", i, i);  
}
```

2. Ввод: функция **getchar()** – возвращает код введенного символа.

Вывод: функция **putchar(char c)**.

3. Для ввода/вывода в файл – функции `getc()`, `putc()`.

4. Функции `ungetc()`, `getch()` – посмотреть самостоятельно!



## Функции для анализа символов `#include <ctype.h>`

Аргумент каждой из них имеет тип `int` и должен представлять собой либо `EOF`, либо `unsigned char`, приведенный к `int`; возвращаемое значение имеет тип `int`. Функции возвращают ненулевое значение (истина), если аргумент `c` удовлетворяет соответствующему условию или принадлежит указанному классу символов, и нуль (ложь) в противном случае.

---

<code>isalnum (c)</code>	ИСТИННО <code>isalpha (c)</code> ИЛИ <code>isdigit (c)</code>
<code>isalpha (c)</code>	ИСТИННО <code>isupper (c)</code> ИЛИ <code>islower (c)</code>
<code>iscntrl (c)</code>	Управляющий символ
<code>isdigit (c)</code>	Десятичная цифра
<code>isgraph (c)</code>	Отображаемый символ, за исключением пробела
<code>islower (c)</code>	Буква нижнего регистра
<code>isprint (c)</code>	Отображаемый символ, в том числе пробел
<code>ispunct (c)</code>	Отображаемый символ, за исключением пробела, буквы или цифры
<code>isspace (c)</code>	Пробел, прогон страницы, конец строки, возврат каретки, табуляция, вертикальная табуляция
<code>isupper (c)</code>	Буква верхнего регистра
<code>isxdigit (c)</code>	Шестнадцатеричная цифра

<code>int tolower(int c)</code>	переводит <code>c</code> в нижний регистр;
<code>int toupper(int c)</code>	переводит <code>c</code> в верхний регистр.

## #define EOF (-1)

Макрос **EOF** – определен в файле `stdio.h` и служит для индикации того, что в потоке больше нет данных. В консоли Windows это сочетание **Ctrl+Z**.

**Пример:** программа для считывания символов с клавиатуры и вывода их на экран. Работает до нажатия Ctrl+Z. [Подробнее](#)

---

```
int main() {  
    int c;  
    while ((c = getchar()) != EOF) {  
        putchar(c);  
    }  
    getch();  
    return 0;  
}
```

При операциях ввода-вывода выделяется область временной памяти (**буфер**), куда и помещаются поступающие символы. Как только поступает специальный сигнал (например, переход на новую строку при нажатии Enter), данные из буфера передаются по месту своего назначения (на экран, в переменную и др.).

# Строки

---

**Строка** – это массив элементов типа char, заканчивающийся символом '\0' (NULL).

```
char A1[20] = { 's', 't', 'r', 'i', 'n', 'g', '\0' };
```

```
char A2[] = { 's', 't', 'r', 'i', 'n', 'g', '\0' };
```

```
char A3[20] = "string";
```

```
char A4[] = "string";
```

```
char * A5 = "string";
```

```
char B[10];
```

## Ввод строки:

**scanf**(" %s", **str**); // до **1**-го пробельного символа

**gets**( **str** ); // ввод до **Enter**

При вводе строки символ **'\0'** добавляется без набора **'\0'** с клавиатуры.

## Вывод строки:

**printf**(" %s ", **str**); // вывод символов до **'\0'**

**puts**( **str** ); // . . . до **'\0'**

# Функции для работы со строками

`#include <string.h>`

---

**char \* strcpy (s, st)** – копирует строку st в строку s, включая '\0'.

**int strcmp (cs, ct)** – сравнивает строки cs и ct;

возвращает <0, если cs<ct;

0, если cs==ct;

и >0, если cs>ct.

**size\_t strlen (cs)** – возвращает длину строки cs .