Целые типы данных

Тип	размер в байтах		диапазон значений
	x86-32	x86-64	
`char`	1	1	зависит от реал.
`signed char`			$[-128; 127]$
`unsigned char`			$[0; 255]$
`int == signed int`	4	4	$[-2^{31}; 2^{31} - 1]$
`unsigned int`			$[0; 2^{32} - 1]$
`short int`	2	2	$[-32768; 32767]$
`unsigned short int`			$[0; 65535]$
`long int`	4	8	зависит от реал.
`unsigned long int`			зависит от реал.
`long long int`	8	8	$[-2^{63}; 2^{63} - 1]$
`unsigned long long int`			$[0; 2^{64} - 1]$

Инициализация и константы

тип имя_переменной = константа

```
char c = 'C'; /* внимание - одинарные кавычки */
int i = 10, j = -10;
unsigned int ui = 10U;
long int li = 10L, lj = -10L;
unsigned long int uli = 10UL;
long long int lli = 10LL, llj = -10LL;
unsigned long long int lli = 10ULL;
```

Восьмеричные и шестнадцатеричные константы

- Восьмеричная константа должна начинаться с 0 (нуля):
  `int oct = 012; /* 10 в десятичной */`
- Шестнадцатеричная константа начинается с 0x (нуль-экс) и содержит символы от A до F, обозначающие цифры от 10 до 15:
  `int hex = 0xA; /* 10 в десятичной */`

Некоторые заголовочный файлы

<limits.h>

Ограничения и параметры переменных целых типов:

- CHAR_BIT — размер char в битах;
- SCHAR_MIN, SHRT_MIN, INT_MIN, LONG_MIN, LLONG_MIN
  SCHAR_MAX, SHRT_MAX, INT_MAX, LONG_MAX, LLONG_MAX
  – минимальные и максимальные возможные значения целых типа signed;
- UCHAR_MAX, USHRT_MAX, UINT_MAX, ULONG_MAX, ULLONG_MAX – максимальные возможные значения целых типа unsigned;

<stddef.h>

Здесь определен size_t – беззнаковый целый тип возвращаемый оператором sizeof(). Используется для указания размеров каких-либо объектов (в том числе массивов) в конкретной реализации.

Битовые операции

Также могут называться, поразрядными, побитовыми или логическими операциями

операция	название	гибрид с =
~	дополнение	унарная операция
&	AND	& =
|	OR	|=
^	XOR	^=
≪	сдвиг влево	≪=
≫	сдвиг вправо	≫=

Внимание

Эти операции можно применять **только** к целым типам **int** или **char**!

Представление целых чисел

- Целые числа без знака представляются в двоичной системе счисления:

4-bit integer

$0101 = 1 * 2^0 + 0 * 2^1 + 1 * 2^2 + 0 * 2^3 = 5$

$11 = 8 + 2 + 1 = 1011$

the max number $= 2^n - 1$; где n – число битов (15 для 4-bit)

Представление целых чисел

- Целые числа без знака представляются в двоичной системе счисления:

4-bit integer

$0101 = 1 * 2^0 + 0 * 2^1 + 1 * 2^2 + 0 * 2^3 = 5$

$11 = 8 + 2 + 1 = 1011$

the max number $= 2^n - 1$; где n – число битов (15 для 4-bit)

- Числа со знаком: старший бит (most significant bit) выделяется для знака (0 – положительный, 1 – отрицательный). Отрицательное число представляются в «дополнительном коде» (two's complement). Берется положительное и
  1. все биты инвертируются ($0 \to 1$; $1 \to 0$),
  2. добавляется 1.

представление числа -5

```
1) 5 = 0101 -> 1010          2) 1010 + 1 -> 1011 = -5
```

Преимущества дополнительного кода:

- Имеется знаковый бит.
- Обратное преобразование будет таким же:

```
1) -5 = 1011 -> 0100          2) 0100 + 1 -> 0101 = 5
```

- Простота сложения:

```
      1011  =   -5          1011  =  -5
      0101  =   +5          0011  =  +3
      ----      --          ----      --
1 <- 0000   =    0          1110  =  -2
```

- Ноль имеет единственное представление

Преимущества дополнительного кода:

- Имеется знаковый бит.
- Обратное преобразование будет таким же:

```
1) -5 = 1011 -> 0100          2) 0100 + 1 -> 0101 = 5
```

- Простота сложения:

```
      1011  =   -5          1011  =  -5
      0101  =   +5          0011  =  +3
      ----      --          ----     --
1 <- 0000   =    0          1110  =  -2
```

- Ноль имеет единственное представление

Недостатки дополнительного кода:

- Неочевиден
- Модули наибольшего и наименьшего чисел различаются:
  the max. positive number $= 2^{(n-1)} - 1$; (7 для 4-bit)
  the min. negative number $= -2^{(n-1)}$; (-8 для 4-bit)

Альтернативное представление целых чисел

Offset binary

- Так же называется **excess-K**: значение числа получается вычитанием из без-знакового заранее фиксированного K
- Применяется для представления показателя чисел с плавающей точкой (IEEE-754) с $K = 2^{(n-1)} - 1$
- Пример для 4-bit: $K = 2^3 - 1 = 7$

0000	–	минимальное число	-7
0001	–		-6
...			
0111	–	ноль	0
...			
1110	–		$+7$
1111	–	максимальное число	$+8$

Битовые операции

~ (дополнение или NOT) инвертирует все биты

```
int i = 3;        /* 0011 */
int j = ~i;       /* 1100 = (-4) */
```

Битовые операции

~ (дополнение или NOT) инвертирует все биты

```
int i = 3;       /* 0011 */
int j = ~i;      /* 1100 = (-4) */
```

& (битовое “И”)

```
int i = 5;       /* 0101 */
int j = 3;       /* 0011 */
int k = i & j;   /* 0001 */
```

Применение – **проверка** состояния индивидуальных битов.

```
/* проверка чётности */
if( j & 0x1 ) {
  printf(" %i is the odd number\n",j);
}
```

| (битовое “ИЛИ”)

```
int i = 5;       /* 0101 */
int j = 3;       /* 0011 */
int k = i | j;   /* 0111 */
```

Применение – **установка** состояния бита в единицу.

| (битовое “ИЛИ”)

```
int i = 5;       /* 0101 */
int j = 3;       /* 0011 */
int k = i | j;   /* 0111 */
```

Применение – **установка** состояния бита в единицу.

^ (исключающее “ИЛИ”, XOR)

```
int i = 5;       /* 0101 */
int j = 3;       /* 0011 */
int k = i ^ j;   /* 0110 */
```

Соответствует булевой функции «сложение по модулю 2» и имеет огромное число применений.

```
/* XOR swap algorithm: меняем местами a=0101 и b=0011 */
a = a ^ b;       /* a = 0110 */
b = b ^ a;       /* b = 0101 */
a = a ^ b;       /* a = 0011 */
```

$\ll$ (сдвиг влево)

Старшие биты исчезают, на место младших записываются нули:

```
unsigned int j = 3;       /* 0011 */
unsigned int k = j << 2; /* 1100  = 12 */
```

`<<` (сдвиг влево)

Старшие биты исчезают, на место младших записываются нули:

```
unsigned int j = 3;       /* 0011 */
unsigned int k = j << 2; /* 1100  = 12 */
```

`>>` (сдвиг вправо)

Младшие биты уходят, на место старших записываются нули:

```
unsigned int j = 3;       /* 0011 */
unsigned int k = j >> 1; /* 0001  = 1 */
```

Результат `E1 << E2` или `E1 >> E2` неопределен если:

- E2 < 0
- E2 $>$ число разрядов E1

Сдвиг вправо и расширение знакового бита

Что будет если сдвигать отрицательные целые?

```
int a = -5;      /* 1011 */
int b = a << 1; /* 0110 = 6 */
/* implementation-dependend: */
int c = a >> 1; /* 0101 = 5 или 1101 = -3 ?? */
int d = a >> 2; /* 0010 = 2 или 1110 = -2 ?? */
```

Стандарт C не определяет точно какой тип сдвига вправо используется

Де-факто (но не гарантированно стандартом):

- **unsigned** – логический сдвиг (заполнение нулями);
- **signed** – арифметический сдвиг (повторение знакового бита)

Применение битовых операций: возведение в степень X^n

Идея: легко вычисляемые степени это $X^1 \quad X^2 \quad X^4 \quad X^8 \quad \ldots$

Представим показатель степени в двоичном виде. Например: 5 = 0101

$X^5 = X \times X^4 \equiv 1 \cdot X \otimes 0 \cdot X^2 \otimes 1 \cdot X^4$

```
double power(double x, unsigned int n) {
   double res = 1.;
   while( n > 0 ) { /* while(n) */
      if( n & 0x1 ) { /* if( n%2 ) */
         res *= x;
      }
      x *= x;
      n >>= 1; /* n /= 2 */
   }
   return res;
}
```

продолжение

```
int main() {
   double x = 3.4;
   unsigned int i = 6;
   double r = power(x,i);
   printf("(%f)^%d = %f log(r)/log(x) = %18.16f\n",
        x, i, r, log(r)/log(x));
   return 0;
}
```

Output:

```
(3.400000)^6 = 1544.804416 log(r)/log(x) = 6.0000000000000000
```

Тип char

- **char** предназначен для хранения символов используя ASCII-кодировку
- размер памяти для хранения – 1 байт (8 бит *de facto*)
- символы от 'A' до 'Z', от 'a' до 'z' и от '0' до '9' идут непрерывно
- символы национальных нелатинских языков (русский, греческий . . .) обычно располагают во второй половине таблицы (128 – 255)
- не стоит использовать ASCII численные значения непосредственно

ASCII Code Chart

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Пример

```
char ch;
for( ch = 'a'; ch <= 'z'; ch++ )
   printf(" %c", ch);
printf("\n");

char c = 'C';
int ic = 65;
printf(" c= %d\n ic= %c\n",c,ic);
```

Output:

```
a b c d e f g h i j k l m n o p q r s t u v w x y z
c= 67
ic= A
```

Расширенный набор целых типов (C99)

Заголовочный файл

```
#include <inttypes.h> // new integer types in C99
```

$N = 8, 16, 32, 64$

Цель	Тип	min	max
Точный размер	int$N$_t uint$N$_t	INTN_MIN 0	INTN_MAX UINTN_MAX
Не менее чем	int_least$N$_t uint_least$N$_t	INT_LEASTN_MIN 0	INT_LEASTN_MAX UINT_LEASTN_MAX
Самые быстрые	int_fast$N$_t uint_fast$N$_t	INT_FASTN_MIN 0	INT_FASTN_MAX UINT_FASTN_MAX

- «Точный размер» – гарантировано имеют N-бит во всех реализациях
- «Не менее чем» – гарантированно не менее N-бит.
- «Самые быстрые» – самые быстрые, которые гарантированно имеют не менее N-бит

. . . продолжение

Цель	Тип	min	max
Наиболее длинные	`intmax_t` `uintmax_t`	`INTMAX_MIN` 0	`INTMAX_MAX` `UINTMAX_MAX`
Для указателей	`intptr_t` `uintptr_t`	`INTPTR_MIN` 0	`INTPTR_MAX` `UINTPTR_MAX`

- «Наиболее длинные» – гарантированно наибольшее количество бит в данной реализации
- «Для указателей» – гарантированно можно хранить указатели

Поддержка компиляторами стандарта C99

- **GCC** – начиная с версии gcc-4.5, C99 практически полностью поддерживается (нет имен переменных в UTF-8)
- **Microsoft Visual C++** – версии до 2012 C99 не поддерживают. В Visual C++ 2013 – частичная поддержка.
- **Clang** – полная поддержка C99.

Спецификаторы класса памяти переменной

static

- создаются один раз и инициализируются нулем
- статические локальные переменные сохраняют своё значение между вызовами функции
- статическая глобальная переменная видна только в том файле, в котором она объявлена

Пример: счётчик вызова функции

```
int fun() {
  static int ncalls=1;
  /* Печатаем сколько раз мы эту функцию вызвали: */
  printf("number of calls %d\n", ncalls++);
  ...
}
```

Пример №2: выполнение подготовительных вычислений

```
double log_gam(double x) {
   static int init_done = 0;
   static double half_ln_2pi;
   static double b2k[9];
   int k;
   if( !init_done ) { /* initialization block */
     init_done = 1;
     half_ln_2pi = 0.5*log(2*M_PI);
     /* Bernoulli numbers */
     b2k[1] = 1./6.; b2k[2] = -1./30.; ...
     /* coeffitients for series */
     for(k = 1; k < 9; k++) b2k[k] /= (double)(2*k*(2*k-1));
   }
   /* calculation of series */
   double sum = 0;
   for(k = 1; k < 9; k++) sum += ...
   return (x-0.5)*log(x) - x + half_ln_2pi + sum;
}
```