Введение в язык программирования C.

Лекция 2

Сергей Леонидович Бабичев

Операции

Операция присваивания

- Имеется *левая часть* операции присваивания
- *l-значения* — то, что может находиться в левой части.
- *переменные* относятся к *l-значениям*.
- *r-значения* — то, что может находиться в правой части.

```
12345 = i;  // Wrong
123 + a = b // Wrong
b = 123 + a // Good;
```

Приведение типов

Преобразование из более длинного целочисленного типа в более короткий производится *отбрасыванием* «лишних» битов.

1. Вещественные типы старше целочисленных.
2. Беззнаковые типы старше знаковых.
3. Длинные типы старше коротких.

Особенности операций

Результаты операций рассматриваются просто как числа с тем же количеством бит. Лишние биты просто отбрасываются.

```
int x = 2000000000, y = 2000000000;
int z = x + y; // z < 0 here.
```

Особенности операций деления / и остатка %

- Если делимое и делитель — неотрицательны, то всё традиционно, `7 / 3 = 2`, а `7 % 3 = 1`.
- Если или делимое или делитель отрицательны, то знак остатка совпадает со знаком делителя. Так в математике и `Python`.
- `Си` использует аппаратные особенности компьютеров, на которых он исполняется, а они могут эти правила не соблюдать. На `Intel/AMD/ARM`

```
int d = -17 / 10; // d <- -1
int e = -17 % 10; // e <- -7
```

- Это не `Python` и не `Pascal`! Операция деления — одна, `/`.
- Для вещественных чисел операции остатка нет.

Побитовые операции

Их немного:

- `&` — побитовое *и*;
- `|` — побитовое *или*;
- `^` — *побитовое исключающее или*;
- `~` — побитовое *не*;
- `<<` — *сдвиг влево*;
- `>>` — *сдвиг вправо*;

Они применяются к каждому из битов в операнде. Всё происходит строго по законам алгебры логики. Если один из операндов короче другого, он расширяется по правилам приведения типов.

Побитовые операции

Операция *побитовое и* имеет следующую таблицу истинности:

```
0 & 0 = 0
0 & 1 = 0
1 & 0 = 0
1 & 1 = 1
```

Операция *побитовое или* имеет следующую таблицу истинности:

```
0 | 0 = 0
0 | 1 = 1
1 | 0 = 1
1 | 1 = 1
```

Примеры побитовых *и* и *или*

```
int ia = 3;        //       00000000 00000000 00000000 00000011
char cb = 5;       //                                  00000101
int ic = -3;       //       11111111 11111111 11111111 11111101
char cd = -5;      //                                  11111011
int ie = ia & cb; // cb -> 00000000 00000000 00000000 00000101
                  // ie  = 00000000 00000000 00000000 00000001
int ig = ic & cd; // cd -> 11111111 11111111 11111111 11111011
```

Для побитовых операций старайтесь использовать беззнаковые типы данных. Избегайте отрицательных значений до тех пор, пока вы точно не будете понимать, как они представляются в компьютере.

Операция исключающего или

```
0 ^ 0 = 0
0 ^ 1 = 1
1 ^ 0 = 1
1 ^ 1 = 0
```

Она обладает потрясающим свойством:

```
int a = 123, b = 555;
int c = a ^ b;
// c^a -> b, c^b -> a.
```

Операция ~(тильда)

Операция `тильда ~`, *побитовое не* для одиночных битов.

```
~0 = 1
~1 = 0
```

Однако не думайте, что если в программе вы напишете ~0, то получите единицу! А что?

Операции сравнения

- Их шесть: >, <, >=, <=, ==, !=
- Два операнда, всё преобразуется к старшему из типов. Результат — 1, если условие истинно или 0, если оно ложно.

> Не путайте операции сравнения (==) и присваивания (=).

```
3 < 5                    // 1
5 == 4                   // 0
```

Сравнение вещественных чисел

- Вещественные числа неточны.

```
double d = 1.0;
int r = (d / 3.0) * 3.0 == d; // r = 1? r = 0?
float f = 1000000000; // f != 10^9
float g = f + 1; //  g == f
```

Сравнение вещественных чисел на равенство принципиально неверно! Все вещественные числа имеют *относительную погрешность*, зависящую от типа. Значения типа `float` имеют погрешность `FLT_EPSILON`, значения типа `double` — `DBL_EPSILON`. Забегая вперёд скажем, что эти константы располагаются в заголовочном файле `math.h`.

Логические операции

Их немного. Это:

- логическое И (`&&`)
- логическое ИЛИ (`||`)
- логическое НЕ (`!`)

Не путайте побитовые операции `&`, `|` и логические `&&`, `||`.
`5 | 3 == 7`, а `5 || 3 == 1`
`3 & 4 == 0`, а `3 && 4 == 1`

Эти операции — *ленивые*. Вычисляется левый операнд, и если оказывается, что результат не зависит от правого операнда, то правый операнд не вычисляется.

Тернарная операция. Приоритеты операций

Она имеет ровно три операнда.

```
x = a > b ? a : b;
```

Приоритеты операций

1. `() [] -> .`
2. `! ~ ++ -- u+ u- u& u* (cast) sizeof`
3. `* / %`
4. `+ -`
5. `<< >>`
6. `< <= >= >`
7. `== !=`
8. `&`
9. `^`
10. `|`
11. `&&`
12. `||`
13. `?:`
14. `= += -= *= /= %= &= |= ^= <<= >>=`
15. `,`

Приоритеты операций

Для группировки операций в выражении в нужном порядке используйте круглые скобки. Используйте их также в случаях, когда имеются хоть малейшие сомнения в порядке исполнения операций в выражении. Операции присваивания исполняются в последнюю очередь.

Первые программы.

Первая завершённая программа

```
01  #include <stdio.h>
02
03  int main() {
04    printf("Hi again\n");
05    return 0;
06  }
```

01. Включить файл из *стандартной библиотеки*.
02. Разделение фрагментов.
03. Определение функции `main`, начало *блока*.
04. Вызов *библиотечной функции* `printf` для вывода текста.
05. Возврат в *вызывающую функцию* с кодом возврата 0.
06. Завершение блока, начатого в строке 03.