

1.2. Типы переменных. Арифметические выражения.

Wednesday, March 11, 2020 23:35

В данном уроке мы рассмотрим различные типы переменных, правила их называния, а также приведение типов переменных (преобразование одного типа в другой, англ. **type casting**). И еще рассмотрим математические операторы и арифметические выражения.

Типы переменных

В первом уроке мы уже успели познакомиться с переменной типа `int` – целочисленное значение (integer). Всего в языке C есть пять основных типов переменных:

- ♦ `int` – целочисленное значение. То есть данный тип переменной неспособен хранить дробные значения. Для вывода в строке используется параметр (**format character**) `%i`
- ♦ `float` – значение с плавающей точкой. Может хранить дробные значения. Можно записывать как в традиционном формате, так и в научном формате: `1.7e4` (что означает 1.7×10^4). Число до буквы `e` называется мантиссой, а число после `e` – экспонентой. Разделитель экспоненты и мантиссы может быть записан в любом регистре: `e` и `E` будут работать одинаково правильно.
Традиционная запись может быть разных форматов: `3.`; `125.8` и `-.0001` являются верными форматами записи.
Для вывода значения в классическом формате используется параметр `%f`. По умолчанию `printf()` отображает 6 чисел после запятой и округляет последнюю цифру.
Для вывода в научном формате используется параметр `%e`. Опять же, отобразится 6 чисел после запятой с округлением последней цифры.
А еще можно предоставить программе выбор, в каком формате отображать, – для этого используется параметр `%g`. Он выбирает между `%f` и `%e` и автоматически убирает неинформативные нули после запятой.
- ♦ `double` – то же самое, что и `float`, но здесь гораздо больше максимально допустимое значение переменной (точность вдвое выше). Обычно под этот тип выделяется 64 бита. Если не указать иначе, по умолчанию в языке C все константы с плавающей точкой имеют тип `double`. Чтобы задать им тип `float`, нужно в конец дописать `f` или `F`, например: `12.5f`.
Для вывода в строку используются те же параметры `%f`, `%e` и `%g`.
- ♦ `char` – обычно используется для хранения символов, например `'a'`, `'0'` или `','`. Следует обратить внимание, что здесь `'0'` не является числом, – это символ и его нельзя подставлять в численную формулу. Кстати `\n` – это символьная константа (ведь перенос строки также является символом). Для вывода в строку используется параметр `%c`
- ♦ `_Bool` был добавлен в стандарте C99. Хранит однобитное значение, т.е. 0 или 1. Сколько под него выделится памяти, зависит от системы. Но обычно гораздо больше, чем 1 бит. Также, дополнительный заголовочный файл `<stdbool.h>` определяет для него псевдоним `bool`, а также макросы `true` (истина) и `false` (ложь). `_Bool` ведёт себя также как и обычный встроенный тип, за одним исключением: любое ненулевое (не ложное) присваивание `_Bool` хранится как единица. Такое поведение защищает от переполнения. То есть, например, после операции `_Bool b = 256` переменная `b` будет равняться 1, т.к. ей было присвоено ненулевое значение [2]. Выводится в строку через `%i`, т.к. 0 и 1 – целые числа.

В языке C любое заданное в коде число, символ или строка является константой. Например, строка `"Hello world!\n"` также является примером константы. Выражения, состоящие только из констант, называются постоянными выражениями (constant expressions). То есть выражение

`128 + 7 - 17`

является постоянным, потому что его значение ни при каких условиях не может измениться. Но если мы введем в это выражение, например, переменную `i` типа `int`, то выражение

128 + 7 - i

уже не будет всегда одинаковым, потому что *i* - переменная и она может меняться по ходу выполнения кода.

Каждый тип переменной, будь это `char`, `int` или `float`, имеет *диапазон значений*, ассоциированный с ним, который зависит от количества памяти, выделяемый под конкретный тип. Количество выделяемой для них памяти зависит от системы, в которой работает программа, т.е. **implementation dependent**.

Например, под тип `int` у 32- и 64-битных процессоров выделяется как минимум 32 бита, а у 16- и 8-битных – обычно 16 бит. Если нужно четко задать, сколько памяти должно быть выделено для переменной, есть типы переменных фиксированной длины, вроде `int32_t` [3].

В языке C помимо десятичного представления числа можно также вывести число в восьмеричном и шестнадцатеричном формате. Для вывода в восьмеричной системе используется параметр `%o`, а для вывода в шестнадцатеричной – `%x`. Чтобы вывести приставку, указывающую на систему счисления, нужно дописать #, т.е. `%#o` припишет 0 к началу числа, а `%#x` припишет 0x. В итоге вывод будет выглядеть примерно так:

```
1      int integerVar;  
2  
3      integerVar = 16772877;  
4      printf ("integerVar is %#x\n", integerVar);
```

Данный код напечатает десятичное значение `integerVar` в шестнадцатеричной системе счисления:

```
integerVar is 0xffef0d
```

Теперь напишем программу, которая выведет известные нам типы переменных:

```
1  #include <stdio.h>  
2  
3  int main(void)  
4  {  
5      int      integerVar = 100;  
6      float    floatVar = 331.79;  
7      double   doubleVar = 8.44e+11;  
8      char     charVar = 'W';  
9      _Bool    boolVar = 50;  
10  
11     printf ("Integer is %i.. Or %#x in hex\n", integerVar, integerVar);  
12     printf ("Float is %f\n", floatVar);  
13     printf ("Double is %e... Or %g\n", doubleVar, doubleVar);  
14     printf ("Char is %c\n", charVar);  
15     printf ("Bool is %i\n", boolVar);  
16  
17     return 0;  
18 }
```

Программа выдаст:

```
Integer is 100.. Or 0x64 in hex  
Float is 331.790009  
Double is 8.440000e+011... Or 8.44e+011  
Char is W  
Bool is 1
```

Внимательный читатель заметил, что вместо 331.79 программа вывела 331.790009. Это вызвано погрешностями вычислений. Вообще, выведенное значение будет зависеть от конкретной системы, на которой код работает (точнее, как на ней реализованы вычисления с плавающей точкой). Некоторые числа с плавающей точкой не могут быть точно описаны в памяти компьютера.

Спецификаторы типов переменных (Type Specifiers)

Мы можем изменять диапазоны значений основных типов переменных при помощи спецификаторов. Рассмотрим их.

- ♦ `long` удлиняет диапазон значений переменной. Например, `long int` расширит диапазон типа `int`. Есть еще `long double` и `long long`. Кстати, если просто написать `long`, без `int`, он автоматически воспримется языком как `long int`.

В большинстве компьютерных систем это бесполезный спецификатор, ничего не меняющий [4].

Если нужно задать тип `long` константе, то для этого используется буква `L` (любого регистра):

```
long int    intVar = 131071100L;
long double doubleVar = 1.234e+7L;
```

Тип `long` выводится в строку буквой `l` – она используется как модификатор для типа `long`: `%li` отобразит `long int` в десятичном формате, `%lx` в шестнадцатиричном, а `%lo` в восьмеричном.

Тип `long long` выводится в строку параметром `%lli`, по аналогии с `long`, только тут две буквы `l`.

Тип `long double` выводится в строку параметром `%Lf`, по аналогии с `long`, только `L` тут большая.

- ♦ `short` укорачивает диапазон переменной типа `int`. Обычно `short int` занимает 16 бит вместо 32. Опять-таки, можно писать просто `short` и компилятор его воспримет как `short int`. Для отображения в строке используется модификатор – буква `b`: `%hi`, `%ho` или `%hx`.

- ♦ `unsigned` создает беззнаковый тип переменной (т.е. без отрицательных значений). Также удваивает верхнюю границу диапазона значений. Кстати, `char` тоже бывает беззнаковым.

Константа типа `unsigned int` обозначается буквой `U`:

```
unsigned int counter = 0x00ffU;
```

То же самое для `long`:

```
unsigned long var = 20000UL;
```

Давайте составим таблицу, которая подведет итог того, что мы написали выше:

Type	Constant Examples	printf chars
char	'a', '\n'	%c
_Bool	0, 1	%i, %u
short int	—	%hi, %hx, %ho
unsigned short int	—	%hu, %hx, %ho
int	12, -97, 0xFFE0, 0177	%i, %x, %o
unsigned int	12u, 100U, 0xFFu	%u, %x, %o
long int	12L, -2001, 0xffffL	%li, %lx, %lo
unsigned long int	12UL, 100ul, 0xffeeUL	%lu, %lx, %lo
long long int	0xe5e5e5e5LL, 5001l	%lli, %llx, %llo
unsigned long long int	12ull, 0xffeeULL	%llu, %llx, %llo
float	12.34f, 3.1e-5f, 0x1.5p10, 0x1P-1	%f, %e, %g, %a
double	12.34, 3.1e-5, 0x.1p3	%f, %e, %g, %a
long double	12.341, 3.1e-51	%Lf, %Le, %Lg

прим.: чтобы точно узнать диапазон переменной или спецификатора, в C предусмотрен интерфейс к свойствам базовых типов [2].

Правила называния переменных

- ◆ Название переменной должно начинаться с буквы или подчеркивания (_).
- ◆ В названии не должно быть символа \$.
- ◆ `int` – зарезервированное слово и так нельзя называть переменную.
- ◆ Название переменной не должно начинаться с цифры.
- ◆ Регистр символов имеет значение. `IntVar` и `intvar` – две разные переменные.
- ◆ Максимальная длина названия переменной – 63 символа, в некоторых случаях – 31.
- ◆ Название переменной должно максимально кратко и точно обозначать её суть.

Целочисленная арифметика и унарный минус (Integer Arithmetic and the Unary Minus Operator).

В языке C для базовых арифметических операций используются стандартные символы: `+`, `-`, `*`, `/`. Также следует помнить про последовательность выполнения операций. Например, в выражении `a + b * c`

сначала будет выполнено умножение и только потом сложение.

Напишем программу, которая вычисляет четыре формулы:

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     int a = 25;
6     int b = 2;
7
8     float c = 25.;
9     float d = 2.;
10
11     printf ("6 + a / 5 * b = %i\n", 6 + a / 5 * b);
12     printf ("a / b * b = %i\n", a / b * b);
13     printf ("c / d * d = %f\n", c / d * d);
14     printf ("-a = %i\n", -a);
15
16     return 0;
17 }
```

Программа выдаст:

```
6 + a / 5 * b = 16
a / b * b = 24
c / d * d = 25.000000
-a = -25
```

Разберем первый результат:

1. Так как операция деления имеет приоритет перед сложением, число `a = 25` сначала делится на 5.
2. У операции умножения также более высокий приоритет перед сложением, потому результат деления `25/5` умножается на 2, давая промежуточный результат 10.
3. В конце проводится операция сложения `6 + 10 = 16`.

Второй и третий результаты чуть поинтереснее. Мы использовали две одинаковые формулы, но результаты оказались разными.

В первом случае мы использовали целочисленные переменные. Посмотрим, что при этом будет происходить. `a / b` даст нецелое значение: $25/2 = 12.5$. Но так как переменные целочисленные, то число после запятой будет проигнорировано и промежуточный результат будет равняться просто 12. Ну а $12 \times 2 = 24$.

Во втором случае мы используем числа с плавающей точкой и потому после деления число после точки сохранится и результат будет правильный: 25.

В последнем случае мы инвертируем знак переменной унарным минусом. Эта операция проводится не между двумя переменными, а над одной переменной, потому минус и называется унарным. Также приоритет унарного минуса выше, чем у других арифметических операций (кроме унарного плюса). То есть в выражении

```
c = -a * b;
```

будет производиться умножение $-a$ на b .

Мы упомянули унарный плюс, но на данном этапе разбирать его не будем. О нем можно почитать в [5].

Оператор сравнения по модулю (modulus operator)

Оператор сравнения по модулю обозначается символом процента (`%`). Рассмотрим, как он работает, написав следующую программу. Кстати, в функции `printf` в текстовой строке нам придется написать знак процента дважды, иначе компилятор его может распознать как команду подстановки переменной (вроде `%i`). А если мы напишем знак процента дважды, компилятор поймет, что мы просто хотим написать знак процента и выведет в строку только один знак процента:

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     int a = 25, b = 5, c = 10, d = 7;
6
7     printf ("a = %i, b = %i, c = %i, d = %i\n", a, b, c, d);
8     printf ("a %% b = %i\n", a % b);
9     printf ("a %% c = %i\n", a % c);
10    printf ("a %% d = %i\n", a % d);
11    printf ("a / d * d + a %% d = %i\n",
12            a / d * d + a % d);
13
14    return 0;
15 }
```

Программа выдаст:

```
a = 25, b = 5, c = 10, d = 7
a % b = 0
a % c = 5
a % d = 4
a / d * d + a % d = 25
```

Теперь рассмотрим полученные результаты.

Как видно, при работе с неотрицательными числами оператор выдает остаток от деления двух чисел. Если мы поделим 25 на 5, то в остатке будет 0, что и видим во второй строке. А $25 / 10$ в остатке дает 5, точно так же $25 / 7$ дает 4, что и подтверждается 3 и 4 строками.

Последнюю строку рассмотрим поподробнее.

Для начала обратим внимание, что строка 11 обрывается на середине и продолжается на следующей строке. Так можно делать, потому что конец команды обозначается символом точки с запятой (`;`), а не концом строки. Так удобно делать, если строка получается длинной. Но стоит помнить, что разрывать строки, например в `printf()`, нежелательно.

Теперь перейдем к формуле. У нас переменные целочисленного типа, потому и деление над ними будет производиться соответственно, – с игнорированием числа после запятой. Деление a / d даст промежуточный результат $25 / 7 = 3$. Его умножение на $d = 7$ даст 21. У оператора модуля такой же приоритет, что и у деления и умножения, поэтому сначала будет выполнена операция $a \% d$ и потом сложение. В итоге $21 + 4 = 25$.

Стоит помнить, что оператор сравнения по модулю работает только с целочисленными переменными.

Приведение типов переменных (Integer and Floating-Point Conversions)

Рассмотрим простейшие способы преобразования целочисленных переменных в вещественные и наоборот. Напишем программу:

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     float f1 = 123.125, f2;
6     int i1, i2 = -150;
7
8     i1 = f1;    // float to int conversion
9     printf ("%g assigned to an int produces %i\n", f1, i1);
10
11    f1 = i2;    // int to float conversion
12    printf ("%i assigned to a float produces %f\n", i2, f1);
13
14    f1 = i2 / 100; // int divided by int
15    printf ("Integer %i divided by 100 produces %f\n", i2, f1);
16
17    f2 = i2 / 100.0; // int divided by float
18    printf ("Integer %i divided by 100.0 produces %f\n", i2, f2);
19
20    f2 = (float) i2 / 100; // type cast operator
21    printf("(float) %i divided by 100 produces %f\n", i2, f2);
22
23    return 0;
24 }
```

Программа выдаст:

```
123.125000 assigned to an int produces 123
-150 assigned to a float produces -150.000000
-150 divided by 100 produces -1.000000
-150 divided by 100.0 produces -1.500000
(float) -150 divided by 100 produces -1.500000
```

Как видим в первой строке, при присвоении целочисленной переменной вещественного значения (**int = float**) числа после запятой просто игнорируются.

Но, если сделать наоборот, – присвоить вещественной переменной целочисленное значение, то ничего не потеряется. Что в принципе очевидно. И подтверждается второй строкой.

Третья строка напоминает нам о правилах целочисленной арифметики: даже если мы присваиваем результат деления целочисленного числа на целочисленное (**int / int**) переменной вещественного типа, нас это не спасет от потери знаков после запятой.

Но, если мы поделим целочисленную переменную на вещественное число, то тут уже будут работать правила арифметики с плавающей точкой и мы не потеряем числа после запятой и получим правильный результат. Что подтверждается четвертой строкой.

Операторы приведения типов переменных (The Type Cast Operator)

Последняя строка нашей программы вводит новое понятие – оператор приведения типа [7], [1]:

```
f2 = (float) i2 / 100;
```

Этот оператор используется для временного преобразования переменной в другой тип, например когда нужно подставить целочисленную переменную в формулу и не потерять точность при делении. Можно также привести в пример приведение в целочисленный тип. Строка

```
(int) 29.55 + (int) 21.99
```

в С будет рассчитана как

```
29 + 21
```

потому что при преобразовании в целочисленный тип будут потеряны числа после запятой.

Комбинирование арифметических операторов с присвоением (Combining Operations with Assignment: The Assignment Operators)

В языке С можно использовать арифметические операторы вместе с присвоением. Их ставят перед знаком присвоения, например: `+=` ; `-=` ; `*=` ; `/=` и `%=`.

Так, в строке

```
count += 10;
```

к переменной `count` будет прибавлено число 10. То есть эквивалентной операцией будет

```
count = count + 10;
```

Для добавления или вычитания единицы есть операторы `++` и `--`:

```
i++; // i = i + 1;  
i--; // i = i - 1;
```

Можно и наоборот:

```
++i; // i = i + 1;  
--i; // i = i - 1;
```

Разница есть, но мы разберем ее позже, в текущих примерах мы ее не заметим.

Также можно применять эти операторы к целым выражениям: строка

```
a /= b + c
```

будет посчитана как

```
a = a / (b + c)
```

С такими операторами быстрее пишется код, он выглядит аккуратнее и иногда быстрее работает.

Комплексные числа

В языке С есть также типы для комплексных чисел, которые называются `_Complex` и `_Imaginary`, использующиеся для работы с комплексными и мнимыми числами соответственно. Их поддержка была добавлена в С99. Подробно рассматривать мы эти типы не будем в данном уроке, а просто их упомянем.

Упражнения

Здесь я приведу свои решения к задачам в конце 3 части [1].

4. Преобразование градусов Фаренгейта в градусы Цельсия:

```
1      float C, F = 27;  
2  
3      C = (F - 32) / 1.8;  
4      printf ("%g degrees F = %g degrees in C\n", F, C);
```

5. Тут программа просто переписывает переменной `d` значение переменной `c` и выводит её. Так что программа выдаст `d = d`.

6. Эта задача напомнила мне о том, что в С нет нативной поддержки возведения в степень. Оператор `^` – это XOR, а не степень. Оператор возведения в степень находится в библиотеке `math.h` и называется `pow()` [8]. Но я решил им не пользоваться и просто умножил нужное количество раз:

```
1      float x = 2.55, res;
```

```
2
3         res = 3*x*x*x - 5*x*x + 6;
4         printf("res = %g", res);
```

7. Здесь просто упражнение на использование на научного формата записи:

```
1      float res;
2
3      res = (3.31e-8 * 2.01e-7)/(7.16e-6 + 2.01e-8);
4      printf("res = %e", res);
```

Кстати, забавное наблюдение. Когда я использовал тип переменной `long double` вместо `float`, я получил неправильный результат. Нужно будет подробнее разобрать этот момент.

8. Здесь снова просто закрепление синтаксиса:

```
1      int next_mul, i = 12258, j = 23;
2
3      next_mul = i + j - i % j;
4      printf("Next largest even multiple of %i for %i is %i.\n",
5             j, i, next_mul);
```

Источники:

1. Stephen Kochan – Programming in C (4th Edition); chapter 3
2. https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%82%D0%B8%D0%BF%D0%BE%D0%B2_%D0%A1%D0%B8
3. <https://en.cppreference.com/w/c/types/integer>
4. <https://docs.microsoft.com/en-us/cpp/cpp/data-type-ranges?view=vs-2019>
5. <https://stackoverflow.com/questions/727516/what-does-the-unary-plus-operator-do>
6. <https://habr.com/ru/post/421071/>
7. https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5_%D1%82%D0%B8%D0%BF%D0%B0
8. <https://stackoverflow.com/questions/18733675/to-the-power-of-in-c>