# 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 С 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` или `%d`.
- `float` – значение с плавающей точкой. Может хранить дробные значения.
  Можно записывать как в традиционном формате, так и в научном формате: `1.7e4` (что означает $1.7 \times 10^4$). Число до буквы е назывантся мантиссой, а число после е – экспонентой. Разделитель экспоненты и мантиссы может быть записан в любом регистре: е и Е 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.

  Стандарт С99 также добавляет комплексные числа: `float _Complex, long double _Complex` и т.п.
- `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 языка С 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 используюттся 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`.

  `char` хоть и используется в основном для хранения символов, ничто не мешает его использовать и для хранения чисел. Под `char` выделяется 8 бит, поэтому его диапазон либо от -128 до 127, либо от 0 до 255 в зависимости от знаковости. В отличии от других переменных, здесь желательно всегда указывать знаковость переменной:

  ```
  signed char sch;
  unsigned char uch;
  ```
- `_Bool` был добавлен в стандарте С99. 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.

То есть, например, если мы хотим гарантированно представить число как `float`, мы напишем:

```
profit = 2150.48f;
```

Если компилятор увидит дробь без указанного типа данных, он автоматически присвоит ей тип `double`, что не всегда нужно.

В языке С 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 С помимо десятичного представления числа можно также вывести число в восьмеричном и шестнадцатиричном формате. Для вывода в восьмеричной системе используется параметр `%o`, а для вывода в шестнадцатиричной – `%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 шестнадцатиричной 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`.

- `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, 50011 | %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-5l | %Lf, $Le, %Lg |

Чтобы точно узнать диапазон переменной или спецификатора, в C есть библиотека `limits.h`.

Также хочу добавить про разницу между спецификаторами `%i` и `%d`. Оба предназначены для работы с типом `int`, и в `printf()` нет разницы, какой спецификатор использовать. Но в `scanf()` разница довольно существенная. Спецификатор `%d` работает **только с десятичными числами**, а `%i` **автоматически определяет систему счисления**. То есть, например, `056` он прочитает как восьмеричное число, а `0x56` как шестадцатиричное. Рекомендуется пользоваться спецификатором `%d`, потому что так легче защититься от ошибок при вводе данных пользователем.

---

### Переполнение целочисленных переменных

Когда мы проводим арифметические операции над переменными, может случиться так, что итоговый результат будет слишком большим или слишком маленьким, чтобы поместиться в допустимые границы значений. Это называется **переполнением**. В случае с переполнением знаковых переменных вроде `int`, поведение программы не определено, но результат вычисления, которое привело к переполнению переменной, определенно будет неверным. Беззнаковые переменные переполняются в ноль. То есть, если у нас есть 16-битный `unsigned int`, равный 65535 (его максимально допустимое значение), то после прибавления 1, результат будет 0.

Про поведение чисел с плавающей точкой можно почитать в [16].

## Правила называния переменных

- Название переменной может начинаться только с буквы или подчеркивания ( _ ), хотя использование в начале подчеркиваний и больших букв в целом нежелательно [10]. Лучше начинать с маленькой буквы.
- В названии не должно быть символа $.
- 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.
- Название переменной не должно совпадать с ключевыми словами языка. Список ключевых слов в C99 приведен в таблице ниже. Разные компиляторы могут иметь дополнительные ключевые слова, вроде `asm`.

| | | | |
|---|---|---|---|
| auto | enum | restrict† | unsigned |
| break | extern | return | void |
| case | float | short | volatile |
| char | for | signed | while |
| const | goto | sizeof | _Bool† |
| continue | if | static | _Complex† |
| default | inline† | struct | _Imaginary† |
| do | int | switch | |
| double | long | typedef | |
| else | register | union | |

†C99 only

## 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:

```c
#include <stdio.h>

int main(void)
{
        int a = 25;
        int b = 2;

        float c = 25.;
        float d = 2.;

        printf ("6 + a / 5 * b = %i\n", 6 + a / 5 * b);
        printf ("a / b * b = %i\n", a / b * b);
        printf ("c / d * d = %f\n", c / d * d);
        printf ("-a = %i\n", -a);

        return 0;
}
```

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 12×2 = 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. Также нельзя применять этот оператор на ноль, т.е. `25 % 0` выдаст ошибку, потому что это равносильно делению на ноль.

## Приведение типов переменных (Integer and Floating-Point Conversions)

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

```c
#include <stdio.h>

int main(void)
{
        float f1 = 123.125, f2;
        int   i1, i2 = -150;

        i1 = f1;    // float to int conversion
        printf ("%g assigned to an int produces %i\n", f1, i1);

        f1 = i2;    // int to float conversion
        printf ("%i assigned to a float produces %f\n", i2, f1);

        f1 = i2 / 100; // int divided by int
        printf ("Integer %i divided by 100 produces %f\n", i2, f1);

        f2 = i2 / 100.0; // int divided by float
        printf ("Integer %i divided by 100.0 produces %f\n", i2, f2);

        f2 = (float) i2 / 100; // type cast operator
        printf("(float) %i divided by 100 produces %f\n", i2, f2);

        return 0;
}
```

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.

---

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
9. https://www.geeksforgeeks.org/operators-c-c/
10. https://stackoverflow.com/questions/25090635/use-and-in-c-programs

11. K.N. King – C Programming: A Modern Approach, 2nd Edition; chapter 2
12. K.N. King – C Programming: A Modern Approach, 2nd Edition; chapter 3
13. K.N. King – C Programming: A Modern Approach, 2nd Edition; chapter 7
14. https://stackoverflow.com/questions/18140331/double-and-float-format-displaying-different-results
15. https://stackoverflow.com/questions/2422712/rounding-integer-division-instead-of-truncating/58568736#58568736
16. What every computer scientist should know about floating-point arithmetic – David Goldberg https://www.itu.dk/~sestoft/bachelor/IEEE754_article.pdf https://docs.oracle.com/cd/E19957-01/806-3568/ncg_goldberg.html
17. GNU G library manual – Integers – https://www.gnu.org/software/libc/manual/html_node/Integers.html
18. https://www.badprog.com/c-type-what-are-uint8-t-uint16-t-uint32-t-and-uint64-t
19. Fixed-size floating point types – https://stackoverflow.com/questions/2524737/fixed-size-floating-point-types
20. Exact-Width Floating-Point typedefs – https://www.boost.org/doc/libs/master/libs/math/doc/html/math_toolkit/exact_typdefs.html

## Упражнения

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

**1.** 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);
```

**2.** Тут программа просто переприсваивает переменной `d` значение переменной `c` и выводит её. Так что программа выдаст `d = d`.

**3.** Эта задача напомнила мне о том, что в C 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);
```

**4.** 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.*

**5.** 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);
```

# Проекты из K. N. King, chapter 2

1. *Write a program that uses printf to display a check mark art on the screen.*

```c
#include <stdio.h>

int main (void)
{
     printf("\n             *\n           *\n");
     printf("          *\n     *   *\n");
     printf("      * *\n       *\n");

     return 0;
}
```

2. *Write a program that computes the volume of a sphere with a 10-meter radius, using the formula v = 4/3πr^3 Write the fraction 4/3 as 4.0f/3.0f.*
3. *Modify the program of Programming Project 2 so that it prompts the user to enter the radius of the sphere.*

```c
#include <stdio.h>

int main (void)
{
     float r;
     printf("Enter radius: ");
     scanf ("%f", &r);
     printf("Sphere volume = %.2f\n", (4.0f / 3.0f) * 3.14159f * r * r * r );

     return 0;
}
```

4. *Write a program that asks the user to enter a dollars-and-cents amount, then displays the amount with 5% tax added.*

```c
#include <stdio.h>

int main (void)
{
     float amnt;
     printf("Enter an amount: ");
     scanf ("%f", &amnt);
     printf("With tax added: $%.2f\n", (amnt + amnt * 0.05f));

     return 0;
}
```

5. *Write a program that asks the user to enter a value for x and then displays the value of the following polynomial: $3x^5 + 2x^4 - 5x^3 - x^2 + 7x - 6$.*
6. *Modify the program of Project 5 so that the polynomial is evaluated using the following formula:*

   *((((3x + 2)x - 5)x - 1)x + 7)x - 6.*

   *Note that the modified program performs fewer multiplications. This technique for evaluating polynomials is known as* ***Horner's Rule****.*

```c
#include <stdio.h>

int main (void)
{
     double x;
     printf("Enter x: ");
```

```
    scanf ("%lf", &x);
    printf("Result: %g\n", (((((3*x + 2)*x - 5)*x - 1)*x + 7)*x - 6));

    return 0;
}
```

Важный опыт – переменные типа `double` в `scanf()` нужно читать как `%lf`, а не как `%f`, как это писалось в первом учебнике. Видимо, тип `double` воспринимается компилятором как `long double`.

**7.** *Write a program that asks the user to enter a U.S. dollar amount and then shows how to pay that amount using the smallest number of $20, $10, $5, and $1 bills.*

```
#include <stdio.h>

int main (void)
{
    int amnt, num20, num10, num5;

    printf("Enter a dollar amount: ");
    scanf ("%i", &amnt);
    num20 = amnt / 20;
    amnt -= num20 * 20;
    num10 = amnt / 10;
    amnt -= num10 * 10;
    num5 = amnt / 5;
    amnt -= num5 * 5;

    printf("$20 bills: %i\n", num20);
    printf("$10 bills: %i\n", num10);
    printf(" $5 bills: %i\n", num5);
    printf(" $1 bills: %i\n", amnt);

    return 0;
}
```

## Проекты из K. N. King, chapter 3

**1.** *Write a program that accepts a date from the user in the form `mm/dd/yyyy` and then displays it in the form `yyyymmdd`.*

```
#include <stdio.h>

int main (void)
{
    int month, date, year;

    printf("Enter a date (mm/dd/yyyy): ");
    scanf ("%d/%d/%d", &month, &date, &year);
    printf("%d%02d%02d\n", year, month, date);

    return 0;
}
```

**2.** *Write a program that formats product information entered by the user.*

```
#include <stdio.h>

int main (void)
{
    int id, month, date, year;
    float price;

    printf("Enter item number: ");
```

```
        scanf ("%d", &id);
        printf("Enter unit price: ");
        scanf ("%f", &price);
        printf("Enter purchase date (mm/dd/yyyy): ");
        scanf ("%d/%d/%d", &month, &date, &year);

        printf("\nItem\tUnit\tPurchase\n    \tPrice\tDate\n");
        printf("%-4d\t$%4.2f\t%02d/%02d/%d\n", id, price, month, date, year);

        return 0;
    }
```

3. *Write a program that breaks down an ISBN entered by the user.*

```
#include <stdio.h>

int main (void)
{
      int gs1, groupID, publCode, itemID, checkDigit;

      printf("Enter ISBN: ");
      scanf ("%d-%d-%d-%d-%d", &gs1, &groupID, &publCode, &itemID, &checkDigit);

      printf("\nGS1 prefix: %d\nGroup identifier: %d\n", gs1, groupID);
      printf("Publisher code: %d\nItem number: %d\n", publCode, itemID);
      printf("Check digit: %d\n", checkDigit);

      return 0;
}
```

## Проекты из K. N. King, chapter 4

1. *Write a program that asks the user to enter a two-digit number, then prints the number with its digits reversed.*
2. *Extend the program in Project 1 to handle three-digit numbers.*
3. *Rewrite the program in Project 2 so that it prints the reversal of a three-digit number without using arithmetic to split the number into digits.*

```
    #include <stdio.h>

    int main (void)
    {
        int d1,d2,d3;

        printf("Enter a three-digit number: ");
        scanf("%1d%1d%1d", &d1, &d2, &d3);
        printf("The reversal is: %d%d%d\n", d3, d2, d1);

        return 0;
    }
```

6. *Write a program to compute the check digit for European Article Number (EAN).*

```
    #include <stdio.h>

    int main (void)
    {
        int d1,d2,d3,d4,d5,d6,d7,d8,d9,d10,d11,d12;
        int sumEven, sumOdd, result;

        printf("Enter EAN: ");
```

```c
    scanf("%1d%1d%1d%1d%1d%1d%1d%1d%1d%1d%1d%1d",
          &d1, &d2, &d3, &d4, &d5, &d6, &d7, &d8, &d9, &d10, &d11, &d12);

    sumEven = d2 + d4 + d6 + d8 + d10 + d12;
    sumOdd  = d1 + d3 + d5 + d7 + d9 + d11;
    result  = 9 - (sumEven * 3 + sumOdd - 1) % 10;

    printf("Check digit: %d", result);

    return 0;
}
```