Язык Arduino основан на ранней версии библиотеки, известной под названием Wiring, дополняющей другую библиотеку — Processing. Библиотека Processing очень похожа на Wiring, но написана на языке Java, а не на C, и используется для соединения компьютера с устройствами на Android через USB. Фактически приложение Arduino, которое вы запускаете в своем компьютере, основано на библиотеке Processing. Если у вас появится желание написать собственный интерфейс для компьютера, чтобы соединить его с платой Arduino, обратите внимание на библиотеку Processing **(www.processing.org)**.

Случайные числа

Кто бы что ни говорил, но компьютеры очень предсказуемы. Однако иногда полезно внести в поведение Arduino элемент непредсказуемости. Например, у вас может появиться желание сконструировать робота, перемещающегося по случайному маршруту в комнате: двигающегося в выбранном направлении случайный промежуток времени, поворачивающего на случайный угол и повторяющего цикл снова. Или создать на основе Arduino игровой кубик, позволяющий получать случайные числа от одного до шести.

Стандартная библиотека Arduino включает необходимую для этого функцию. Она называется `random`. Функция `random` возвращает значения типа `int` и может принимать один или два аргумента. Если ей передать только один аргумент, она вернет случайное число в диапазоне от 0 до значения аргумента –1.

Версия с двумя аргументами возвращает случайные числа в диапазоне от первого аргумента (включительно) до второго минус единица. То есть вызов `random(1, 10)` вернет случайное число в диапазоне от 1 до 9.

Скетч 7-01 выводит в монитор последовательного порта числа от 1 до 6:

```
// sketch 7-01
void setup()
{
  Serial.begin(9600);
}
void loop()
{
  int number = random(1, 7);
  Serial.println(number);
  delay(500);
}
```

Если выгрузить скетч в плату Arduino и открыть монитор последовательного порта, можно увидеть картину, напоминающую рис. 7-1.

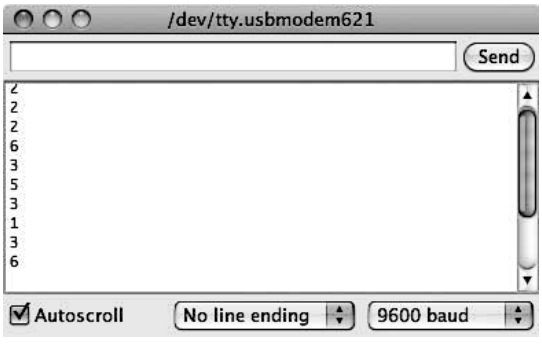

Рис. 7.1. Случайные числа

Если попробовать запустить скетч несколько раз подряд, вы, вероятно, будете удивлены тем, что каждый раз скетч генерирует одну и ту же последовательность «случайных» чисел.

В действительности эти числа не являются случайными — их часто называют *псевдослучайными*, потому что они имеют случайное распределение. То есть если запустить скетч и собрать миллион чисел, вы увидите примерно одинаковое количество единиц, двоек, троек и других чисел. Но числа не являются случайными в смысле их непредсказуемости. В действительности микроконтроллер не в состоянии генерировать по-настоящему случайные числа без вмешательства извне.

Вы можете организовать такое вмешательство, *инициализируя* генератор случайных чисел, чтобы сделать последовательность чисел менее предсказуемой. Но в этом случае можно задать лишь начальное

значение для последовательности. Если задуматься, несложно прийти к выводу, что нельзя использовать random для инициализации генератора случайных чисел. На практике обычно используется трюк, основанный на том факте (обсуждавшемся в предыдущей главе), что «висящий в воздухе» аналоговый ввод подвержен внешним помехам. То есть значение, прочитанное с аналогового входа, можно использовать для инициализации генератора случайных чисел.

Функция, делающая это, называется randomSeed. Скетч 7-02 демонстрирует, как добавить немного случайности в работу генератора случайных чисел:

```
// sketch 7-02
void setup()
{
  Serial.begin(9600);
  randomSeed(analogRead(0));
}
void loop()
{
  int number = random(1, 7);
  Serial.println(number);
  delay(500);
}
```

Попробуйте нажать кнопку **Reset** на плате несколько раз. Вы увидите, что каждый раз после этого генерируется иная последовательность случайных чисел.

Такой способ получения случайных чисел нельзя использовать для розыгрыша лотереи. Чтобы получить их более качественную последовательность, необходима специальная аппаратура, которая опирается на случайные события, такие, например, как интенсивность космического излучения.

Математические функции

В скетчах для Arduino редко возникает необходимость в математических вычислениях сложнее элементарной арифметики. Если это потребуется, к вашим услугам имеются математические функции. В следующей таблице перечислены наиболее интересные из них:

Функция	Описание	Пример
abs	Возвращает абсолютное значение аргумента	abs(12) вернет 12 abs(-12) вернет 12
constraint	Ограничивает значения заданным диапазоном. Первый аргумент — ограничиваемое значение, второй — начало диапазона допустимых значений, третий — конец диапазона	constrain(8, 1, 10) вернет 8 constrain(11, 1, 10) вернет 10 constrain(0, 1, 10) вернет 1
map	Отображает число из одного диапазона в другой диапазон. Первый аргумент — число для отображения. Второй и третий аргументы определяют исходный диапазон, а последние два — конечный. Функция может пригодиться для отображения значений, получаемых с аналоговых входов	map(x, 0, 1023, 0, 5000)
max	Возвращает наибольший из двух аргументов	max(10, 11) вернет 11
min	Возвращает наименьший из двух аргументов	min(10, 11) вернет 10
pow	Возвращает первый аргумент, возведенный в степень, определяемую вторым аргументом	pow(2, 5) вернет 32
sqrt	Возвращает корень квадратный для аргумента	sqrt(16) вернет 4
sin, cos, tan	Тригонометрические функции. На практике используются редко	—
log	Может использоваться для вычисления температуры на основании значений, полученных от логарифмического терморезистора, например	—

Операции с битами

Бит — это единица измерения двоичной информации, он может принимать значение 0 или 1. Слово *bit* является сокращением от англ. *binary digit* — двоичная цифра. В большинстве случаев вы будете использовать переменные типа `int`, фактически состоящие из 16 бит.

Применять такие переменные для хранения логических значений истина/ложь (1 или 0) кажется расточительством. В действительности, если вы не испытываете нехватки памяти, такое расточительство является меньшим злом, чем создание сложного для понимания программного кода, выполняющего операции с битами. Но иногда бывает выгодно иметь возможность упаковывать информацию более плотно.

Каждый бит в переменной типа `int` имеет десятичное значение, и вы можете получить полное значение переменной, складывая десятичные значения, соответствующие битам, равным 1. Так, на рис. 7.2 показано двоичное представление десятичного числа 38. Ситуация немного осложняется при работе с отрицательными числами, но это происходит, только когда самый старший (самый левый) бит равен 1.

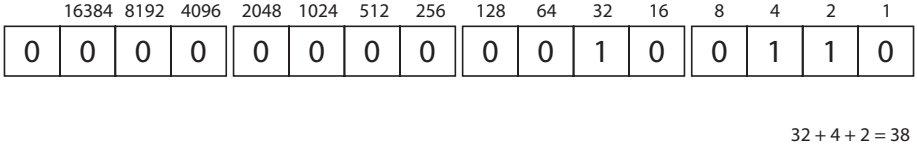

Рис. 7.2. Число типа int

Десятичные значения плохо подходят при работе с битами в уме. Очень сложно представить себе, какие биты установлены в десятичном числе, таком как 123. Поэтому программисты часто пользуются *шестнадцатеричной* системой счисления, то есть системой счисления с основанием 16. В этой системе счисления кроме десятичных цифр от 0 до 9 используются дополнительные шестнадцатеричные цифры от A до F. То есть каждая цифра представлена четырьмя битами. В следующей таблице приводятся соответствия между десятичными, шестнадцатеричными и двоичными представлениями чисел от 0 до 15:

Десятичное	Шестнадцатеричное	Двоичное
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110