1811/2807/7001ICT
Programming Principles

School of Information and Communication Technology
Griffith University

Trimester 1, 2024

3 Numbers

In this section we will discuss some of what programmers need to know about numbers.

Most programs have to use numbers somewhere.

The notations used in this section are mathematical, rather than those used in programming languages.

3.1 Kinds of numbers

3.1.1 Natural numbers

The earliest kinds of numbers people used were the kinds of numbers you need to count things, the *counting* numbers.

$$1, 2, 3, ...$$

Counting numbers have been with us a long time, longer than we have been classifiable as *Homo Sapiens*, since there are animals that can count (up to a few anyway), such as primates and birds.

At first, we had no concept of zero.

The counting numbers are also known as *cardinal* and *natural* numbers.

Most mathematicians and computing scientists would also include zero in the natural numbers.

$$0, 1, 2, 3, ...$$

There is an infinite number of them.

3.1.2 Integers

All of the positive and negative whole numbers and zero are together known as the *integers*.

$$..., -3, -2, -1, 0, 1, 2, 3, ...$$

3.1.3 Rationals

The *rational* numbers are all the numbers you can represent as the ratio (or fraction) of two whole numbers; e.g.

$$-1 = \frac{-1}{1}, \quad 0 = \frac{0}{1}, \quad 0.5 = \frac{1}{2}, \quad 3.333... = \frac{10}{3}$$

3.1.4 Irrationals

There are numbers that can't be represented as the ratio of two whole numbers; e.g.

$$\sqrt{2}, \quad e, \quad \pi$$

3.1.5 Reals

The *real* numbers are all of the rationals and irrationals.

3.1.6 Continuous versus discrete

To pick the kind of numbers needed to represent or solve a problem, the most important choice is between:

continuous values – things that might have fractional values or are smoothly varying – things that can be measured: weight, length, speed.

Use reals.

discrete values – things that can only have whole number values – counting things.

Use naturals or integers.

3.2 Arithmetic operations

3.2.1 Addition

$$3 + 4 = 7$$

3.2.2 Subtraction

$$4 - 3 = 4 + (-3) = 1$$

$$3 - 4 = 3 + (-4) = -1$$

3.2.3 Multiplication

$$4 \times 3 = 4 + 4 + 4 = 3 + 3 + 3 + 3 = 12$$

$$4 \times (-3) = (-3) + (-3) + (-3) + (-3) = -12$$

3.2.4 Division

$$12 \div 4 = 3$$

$$12 \div 5 = 2 \text{ or } 2.4?$$

The answer to the second example can be 2, if we are only considering whole numbers.

Sometimes we use a / instead of ÷.

$$12/5 = 2$$

Sometimes we are even more precise about indicating that we are doing integer division. The word div always means integer division.

$$13 \text{ div } 5 = 2$$

3.2.5 Modulo

We all know that there is a bit left over in an integer division like the one above.

At school we might have said:

$$13 \operatorname{div} 5 = 2 \operatorname{remainder} 3$$

We can compute the remainder with the *modulo* operation, mod.

$$13 \bmod 5 = 3$$

3.2.6 Powers

$$3^2 = 3 \times 3 = 9$$

$$3^3 = 3 \times 3 \times 3 = 27$$

$$3^1 = 3$$

$$3^0 = 1$$

3.2.7 Precedence

A mathematical expression with multiple mathematical operations in it, might have several interpretations.

$$3 + 4 \times 5 = 35 \text{ or } 23?$$

We resolve this with an order of precedence, that is what should we do first?

precedence	*operations*	*example*
highest	*unary* + and −	$2 \times -3 = 2 \times (-3) = -6$
	$\times$, div, $\div$, /, mod	$3 + 4 \times 5 = 3 + (4 \times 5) = 23$
lowest	*binary* + and −	

We can override this conventional order of precedence using parentheses.

$$(3 + 4) \times 5 = 35$$

3.3 Bases

3.3.1 Decimal

We usually count in decimal, that is in *base 10*.

Not every culture in history has done this. In ancient Babylon they counted in base 60. To this day there are 60 minutes to the hour, and 60 seconds to the minute.

Decimal (and Babylonian, but not Roman) numbers are *positional*, that is we have only a few symbols for numbers and the value of the symbol depends on its position in the number.

$$123 = 1 \times 10^2 + 2 \times 10^1 + 3 \times 10^0 = 100 + 20 + 3$$

Decimal numbers are written using the 10 digits $0, 1, 2, ..., 9$.

3.3.2 Binary

Binary numbers, that is numbers in base 2, are written using digits 0 and 1.

They are used in computing a lot, because they are easy to represent in an electronic circuit. A switch that is on can represent 1, and one that is off can represent 0.

Binary numbers are positional, like decimal numbers.

$$1010 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 8 + 0 + 2 + 0 = 10$$

3.3.3 Other bases

In computing we sometimes use bases 8 (digits 0, 1, ..., 7) and 16 (digits 0, 1, ..., 9, A, B, ..., F), because numbers in these bases are shorter than binary numbers and very easy to convert to and from binary numbers.

3.4 Whole numbers in computers

Whole numbers are stored in computers in binary format.

They are saved by setting some number of switches to on and off.

There can only be a finite number of these switches per number, as there is only a finite number of switches in any given computer's memory.

That means a computer can not store natural numbers or integers (there are infinitely many of them) – only subsets of them.

How big that subset is, depends on how much memory you want to use.

3.4.1 Bits, bytes, words

The word *bit* is derived from *bi*nary digi*t*.

A bit is one binary digit, that is a 0 or a 1.

A bit can save a useful amount of information: yes or no; true or false.

A few bits in a row can save a number. For example, with three bits we can count as follows: 000, 001, 010, 011, 100, 101, 110, 111.

That is we can count from 0 to 7 with three bits.

That is there are 8 different numbers you can express with 3 bits. It is no coincidence that $8 = 2^3$.

We usually choose a number of bits per binary number that is itself a power of 2.

A *byte* is 8 bits.

With a byte, you can represent the numbers 0 to 255. $255 = 2^8 - 1$.

To count different things we need different numbers of bits. For most counting jobs as few as 16 bits are needed.

As time has progressed, computer hardware has been able to do arithmetic operations with bigger and bigger binary numbers in one step.

For example an Apple 2 could only perform operations on bytes, but most PCs now can perform operations on 64 bits at a time.

A *word* is a term used to describe a binary number of some number of bits, usually the maximum that a given computer can handle in one step.

3.4.2 Negative numbers

To store a negative number in the binary memory of a computer we have to use a bit to indicate that a number is positive or negative.

Usually this is the left-most bit.

For example, in a 16-bit word:

1001001001001001	must be a negative number; and
0111111111111111	is the biggest positive number you can store.

There are a couple choices as to how exactly negative numbers are represented.

The commonest is two's complement.

3.5 Fractional numbers in computers

In a computer, to represent a fractional number, we store the best approximation to a real number that we can with a floating point representation using a finite number of bits.

Single precision uses 32 bits. *Double precision* uses 64 bits.

The bits are grouped into 4 parts:

- a sign bit;
- the *mantissa*;
- the *exponent*; and
- the sign of the exponent.

The number is represented like scientific notation: $\pm mantissa \times 10^{\pm exponent}$.

The exponent lets us represent very small or very large numbers.

There are some special patterns of bits reserved for some special values:

- infinity; and
- Not a Number (), the result you can get if a computation can't return a sensible result.

Single precision can only approximate numbers to about seven decimal digit's worth of precision. This is not accurate enough to calculate the average person's income to the cent.

Unless the computer has very little memory or you need to store a very large number of numbers, double precision should be used.

Section summary

This section covered:

- the kinds of whole numbers (naturals, integers, rationals, irrationals, reals);
- arithmetic operations (especially modulo);
- precedence of operators;
- representing numbers in different bases;
- how whole numbers are represented in computers (bits, bytes, words);
- how fractional numbers are represented in computers.