Unit 4: Types

Learning Outcomes:

After this unit, students should:

- understand the concept of *type* as a way to interpret the meaning of bits in the memory;
- be able to calculate the number of different possible values that can be represented by a given number of bits;
- understand that different types require a different number of bits for representation;
- understand that not all real numbers can be represented in a computer;
- be aware that real numbers may not be represented precisely in a computer, and so real numbers should never be used as a type of an integer value;
- be aware that C is a statically-typed language, in contrast to Python and Javascript, which are dynamically-typed;
- be aware that the type for all C variables and functions must be declared;
- be aware that choosing the wrong type could lead to an incorrect implementation of an algorithm.

Bits and Bytes

Recall from Unit 1 that machine code and data manipulated by machine code are all stored as a sequence of 1s and 0s in the memory. Each unit of either 1 or 0 is known as a *bit*. 8 bits form a *byte*.

Remember from Unit 2 that a variable is a memory location that stores a value, as a sequence of bits. The bits stored in the memory has no meaning by itself. It has to be interpreted by the machine code. Does a sequence of 1s and 0s represent an integer? A pixel of an image? A sound sample in an audio clip? A month? As a programmer, we have to tag the variable with its *type*, so that the machine code knows how to interpret the sequence of bits. In addition, the type also tells the machine code, how many bits "belong" to this variable. The number of bits of a type is also known as the *size* of a type.

a variable needs a type to be interpreted correctly
like opening image file in notepad

The size of a type determines how many different values a variable of that type can hold. For instance, a type of one bit can only hold two possible values (e.g., `0` or `1`, `true` or `false`, `black` or `white`). A type of two bits can hold four values, represented as `00`, `01`, `10`, `11`. In general, a type of k bits can hold 2^k values.

math

Integers

To represent integers, a type of 8 bits can represent 256 different values. If the type only represents non-negative integers (called `unsigned`), then it can hold any value between 0 to 255. If it represents both positive and negative integers (called `signed`), it can hold any value between -128 to 127. Depending on the needs (how much memory we have, how big is the number we need to represent), programmers have to decide on the size of the type used for a variable in their programs. With 64 bits, a signed integer can store any value between -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807. This might look big enough for you -- but we can't even fit the results from

2^(8-1) [half of the range]

factorial(21) here!
We have to go to 128 bits to represent larger integers.

Characters

All comparing to a lookup table

To represent a character (i.e., a symbol representing a letter, number, punctuation, etc), we use 8 bits for English characters. The ASCII standard maps 127 binary sequences to common characters you can type on your keyboard (including 0-9, A-Z, a-z, all sorts of punctuations: <>?:"{}!@#$%^&*()_+-=[]\';/., and special characters like return and escape). The Unicode standard uses up to 32 bits per character, allowing many non-English letter characters (e.g., Emoticons, Braille, Mahjong Tiles) to be represented.

UTF-8

Real Numbers

avoid working directly with them
- due to chance of precision error
- try to only use them as return values

For real numbers, because we need to represent a huge number of possible values, we normally use 32 bits or 64 bits. In CS1231, you will learn that there are uncountably many possible real numbers. But, no matter how many bits we use, we can only represent a finite number of possible values for real numbers. Here, we run into a limit of digitizing information into 0s and 1s -- we can never represent all possible real numbers in a computer! Because of this, programs that manipulate real numbers leads to weird answers (such as 3.1 + 3.2 becomes 3.30000000000000003) and so, we have to take special care when dealing with real numbers when writing programs (not just in C, but in many other languages as well). Hence, *if we only expect a variable to hold an integer, we should not choose a type that represents real numbers.*

The details of how a sequence of bits can represent integers and real numbers (both positive and negative numbers) are covered in CS2100.

You should have also noticed that in the discussion above, we talk about types that are 8, 16, 32, 64, 128 bits -- all power of twos. We do not have a type of size, say, 41 bits. The reason for this has to do with how memory location is addressed. This will again be explained in CS2100.

Type Declaration

In C, which we will use in CS1010, we have to associate *every* variable with a type, and once a variable is *declared* with a type, the type cannot be changed.

This behavior is known as static typing. Some programming languages, such as Javascript and Python, are dynamically typed. The type of a variable may change depending on the value the variable is assigned to.

When we write a function, we have to declare the types of each of the parameters and the return value as well.

declaring a list

Take the function $mean(L, k)$ as an example. We have said that L is a list[1] of integers. So each element in L should have an integer type (how many bits to use will depend on the range of numbers we want the program to possibly handle and how frugal we are about memory usage). What about k? k refers to the number of elements in L, so it has to be an integer. As for the value returned by $mean(L, k)$, even when the inputs are all integers (and thus, the sum is an integer), the mean value can be a real number. So we should choose a type that represents a real number for the return value.

elements in L are integer, thus sum is integer

division has occurred to find mean thus need to change type for return

> ⚠ **Importance of Type**
>
> Choosing the wrong type to represent a variable can lead to buggy code. Suppose we say that $mean(L, k)$ returns an integer, then when we call *mean* on the input `1 2 3 4`, we will get `2` as the answer, instead of `2.5` as it should.

---

1. We have not talked about how to represent a list yet. I will do that in a later unit. ↵