ECE 131 – Programming Fundamentals

Module 2, Lecture 2: Data Types, Variables and Expressions – Integer and Floating-point Types

Dr. Daryl Lee

University of New Mexico

Number Systems

- Before talking about data types, it is important to first understand number systems.
- We live in a decimal, or base 10, world (probably due to the fact that some mathematician hundreds of years ago had 10 fingers), but computers are constructed around a binary (base 2), number system.
- Thus, we need to understand how to move back and forth between these two number systems.
- E.g., there are various reasons why you might need to determine why the bits in the following 16-bit number are set to 0 or 1:

0	0	1	0	1	0	1	1	0	1	1	1	0	0	1	1

- The terms radix and base are used interchangeably in computing.

THE UNIVERSITY of NEW MEXICO

Number Systems

- Indeed, we will see that the octal (base 8) and hexadecimal (base 16) numbers systems are also very important in computing.
  Note:
  - The base for binary numbers is 2^1, i.e., the numbers 0 and 1. It requires one bit to store one binary number.
  - The base for octal numbers is 2^3, i.e., the numbers $0-7$. It requires three bits to store any one octal number.
  - the base for hexadecimal numbers is 2^4, the numbers $0-9$, followed by $A-F$. It requires 4 bits to store any one hexadecimal number.

  Notice also that these bases are all powers of 2.
- In computing, 8 bits are referred to as a byte and 4 bits are referred to as a nibble, sometime spelled nybble.

THE UNIVERSITY of NEW MEXICO

Number Systems

Given a number d_a, in base a, with digits

$$d_a = d_a[n]d_a[n-1]\ldots d_a[2]d_a[1]d_a[0],$$

the conversion of this number to base $b > a$ is achieved by using the formula:

$$\sum_{i=0}^{n}(d_a[i] \cdot a^i),$$

where the arithmetic in this equation is conducted in base b.

Ex: Convert 1101_2 to base 10.

$$\begin{aligned}(1 \cdot 2^0) + (0 \cdot 2^1) + (1 \cdot 2^2) + (1 \cdot 2^3) &= 1_{10} + 0_{10} + 4_{10} + 8_{10} \\ &= 13_{10}.\end{aligned}$$

THE UNIVERSITY of NEW MEXICO

Ex: Convert 27_{10} to hexadecimal (base 16).

$$\begin{aligned}(2 \cdot 10^1) + (7 \cdot 10^0) &= 14_{16} + 7_{16} \\ &= 1B_{16}.\end{aligned}$$

Note that in hexadecimal, you count like:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F,
10, 11, 12, 13, 14, . . . , 19, $1A, 1B, 1C, 1D, 1E, 1F$,
20, 21, 22, 23, 24, . . . , 29, $2A, 2B, 2C, 2D, 2E, 2F, \ldots$

and you can see that $F_{16} = 15_{10}$ and $10_{16} = 16_{10}$.

If instead, we're trying to convert from a number d_a to a number in base b, where $b < a$, we successively divide d_a by b. Denote the number we're trying to covert to as as c_b. Then,

- Let d'_a denote the dividend and r' the remainder of d_a/b. Set c_0, the lowest-order digit, to r'
- Next compute d'_a/b, and set c_1, the second-lowest-order digit, to r'', the remainder of this division.
- Continue in the fashion until the dividend is reduced to 0.

Ex: Convert 13_{10} to binary.

$$\begin{aligned} 13/2 &= 6 \text{ with a remainder of } 1 \rightarrow c_0 = 1, \\ 6/2 &= 3 \text{ with a remainder of } 0 \rightarrow c_1 = 0, \\ 3/2 &= 1 \text{ with a remainder of } 1 \rightarrow c_2 = 1, \\ 1/2 &= 0 \text{ with a remainder of } 1 \rightarrow c_3 = 1. \end{aligned}$$

Thus, $13_{10} = 1101_2$.

Here's another way to convert 13_{10} to binary :

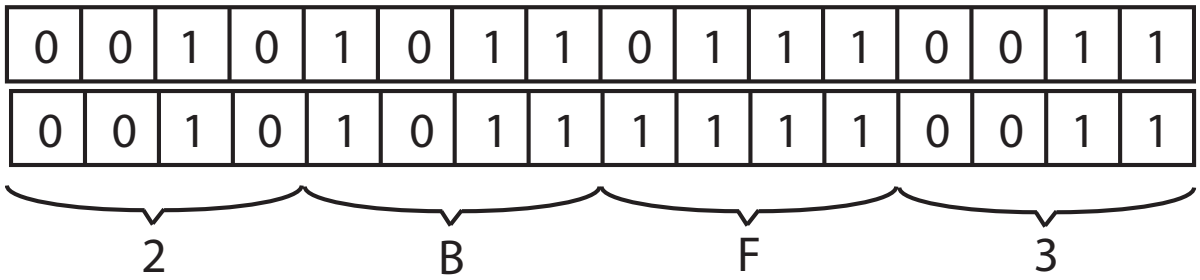
- First, list the powers of 2:

$$\begin{aligned} 2^0 &= 1 \\ 2^1 &= 2 \\ 2^2 &= 4 \\ 2^3 &= 8 \\ 2^4 &= 16 \\ &\vdots \end{aligned}$$

- Since $2^4 = 16 > 13$, but $2^3 = 8 < 13$, we know the $d_2[4] = 0$ and $d_2[3] = 1$. Now, this covers 8 of the 13, and we have 5 leftover. Thus, since $2^2 = 4 < 5$, we know that $d_2[2] = 1$, and now we've covered 12 of the 13, with 1 leftover. Since $2^1 = 2 > 1$ and $2^0 = 1$, we have $d_2[1] = 0$ and $d_2[0] = 1$.

Number Systems - Uses

- The word size of a particular computer architecture refers to the number of bits it uses for its registers and memory locations. E.g., common sizes are 8-, 16-, 32-, and 64-bit architectures.
- It's common to use hexadecimal notation as a compact way to communicate what bits are stored in a word.
  Ex:



Note: $0010101111110011_2 = 2BF3_{16}$

THE UNIVERSITY of NEW MEXICO

Data Types

- Most programs involve the manipulation of data.
- This data usually has a type associated with it.
- More specifically, a data type is specified by determining the set of possible values and basic operations that can be applied to data of the given type.
- Most programming languages provide some data types as part of their specification. These are referred to as built-in data types.
  Ex: Most languages automatically support the notion of an integer data type, i.e., a data type that can store integer numbers ($\mathbb{Z}$), and a floating-point data type, i.e., a data type that can store real numbers ($\mathbb{R}$).
- In addition, programming languages typically provide some way for users to create their own data types. These are referred to as user-defined data types.

THE UNIVERSITY of NEW MEXICO

Integer Data Types

- In mathematics the integer domain, $\mathbb{Z}$, is infinite. A computer can't represent an infinite number of integers, it can only represent some interval, or range, of $\mathbb{Z}$.
- The range of values that an integer data type can represent is determined by the number of bits used by the data type.
  Ex: Consider a 4-bit integer data type –

0_{10} in binary:	0	0	0	0
1_{10} in binary:	0	0	0	1
2_{10} in binary:	0	0	1	0
15_{10} in binary:	1	1	1	1

  So we can represent the range $[0, 15] \subset \mathbb{Z}$ using 4 bits. I.e., $2^4 = 16$ numbers can be represented using 4 bits.
- Using n bits, the interval would be $[0, 2^n - 1]$.

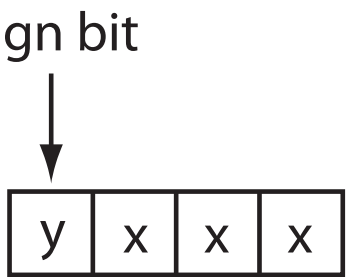
THE UNIVERSITY of NEW MEXICO

Integer Data Types

- Notice in the previous 4-bit example that the rightmost bit was changing most rapidly as we counted from 0 – 15, i.e., $(0000, 0001, 0010, 0011, \ldots, 1110, 1111)$.
- In this case, the rightmost bit is referred to as the least significant bit (LSB), and the leftmost bit is referred to as the most significant bit (MSB).

Integer Data Types

- In the previous example, we considered only positive integers. In programming, these are referred to as unsigned integers.
- To represent both positive and negative integers, referred to as signed integers, a sign bit is added to the previous representation.
  I.e., with 4 bits, and the MSB as the sign bit:



  y = 0, positive integer
  y = 1, negative integer
  x x x = 000, 001, 010, 011, 100, 101, 110, or 111
- We can still represent 16 integers, but now the range is $[-8, 7]$.
- Using n bits, the interval would be $[-2^{n-1},\ 2^{n-1} - 1]$.

THE UNIVERSITY of NEW MEXICO

Signed Integer Data Types

- Nowadays, it's actually not very common to store signed integers in format just described.
- One's complement coding and two's complement coding more common.
- These formats have some advantages in terms of implementing arithmetic circuits.
- Specifically, with two's complement numbering, the addition and subtraction circuitry does not have to explicitly examine the sign bits of two numbers before deciding whether addition or subtraction should be used.
- However, with these coding schemes, the range is unchanged, and we won't consider them further in this class.

Integer Data Types in C

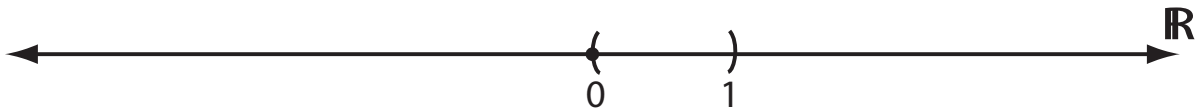
- In C, there are three main built-in integer types:
  - `char` – an 8-bit integer, often used to store ASCII character values, $[-128, 127]$.
  - `int` – uses at least 16 bits, $[-2^{15},\ 2^{15}-1]$.
  - `_Bool` – used to store either a 1 (true) or a 0 (false), a bit.
- In addition, modifiers may appear immediately before a type name. Available modifiers include: `short`, `long`, `long long` and `unsigned`.
  - `short int` – uses at least 16 bits, $[-2^{15},\ 2^{15}-1]$, typically half the size of an `int`.
  - `long int` – uses at least 32 bits, $[-2^{31},\ 2^{31}-1]$.
  - `long long int` – uses at least 64 bits, $[-2^{63},\ 2^{63}-1]$.
  - `unsigned char` – uses 8 bits, $[0, 255]$
  - `unsigned int` – uses at least 16 bits, $[0, 65535]$.
- C program snippet:

```
char x;
int number;
unsigned int counter;
```

THE UNIVERSITY of NEW MEXICO

Real Data Types

- Although the integer domain $\mathbb{Z}$ is infinite, over any range of $\mathbb{Z}$ that does not include $-\infty$ or $+\infty$, there are a finite number of integers.
- Thus, once we picked the number of bits associated with an integer type, every integer in the associated range could be represented *exactly*.
- In mathematics the real domain, $\mathbb{R}$, is also infinite. Furthermore, any interval of $\mathbb{R}$ is also infinite.
  E.g.,

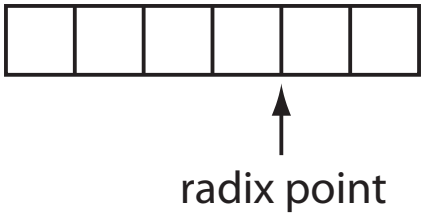


  There are an infinite number of real numbers in the interval $(0, 1)$.
- Thus, we cannot exactly represent every number in a given range of $\mathbb{R}$.

THE UNIVERSITY of NEW MEXICO

Real Data Types

- Consider the situation where we have 6 bits, and let us fix the decimal point (or more generally the radix point, which applies to any base) between the 2nd and 3rd bits:



- Assuming an unsigned number, then since there are 4 bits to the left of the radix point, the range of the numbers that can be represented is the interval $[0, 16)$.
- However, with 6 bits, only 2^6 numbers in this range can actually be represented.
- Because there are 2 bits to the right of the radix point, we can only represent three points between any successive whole numbers in this range.
  E.g., 12.0, 12.25, 12.5, and 12.75.

THE UNIVERSITY of NEW MEXICO

Real Data Types

- Assuming we space the points between two whole numbers equally, the precision of a number will be 0.25.
  E.g., the number 12.472 would need to be rounded to 12.5.
- Because the radix point never changes in this representation, it is referred to as a fixed-point number representation.
- In a floating-point number representation, the radix point is allowed to change. The real number is represented by storing the significand (also called the mantissa) and the signed integer exponent.
- You can think of this as a using scientific notation to store the number.
  E.g., $1278.021 = 1.278021 \times 10^3$
  significand $= 1278021$
  exponent $= 3$
  Note that there is an implied decimal point in this representation.

THE UNIVERSITY of NEW MEXICO

Floating-point Data Types

- By allowing the radix point to float, storage of numbers over a wider range of $\mathbb{R}$ is supported.
- The two main floating-point types in C are:
  - `float` – a standard floating-point number, guaranteed to contain at least 6 digits of precision.
  - `double` – a double precision floating-point number, guaranteed to contain at least 10 digits of precision.
- The variable modifier `long` can be added:
  - `long float` – this is a synonym for `double`.
  - `long double` – guaranteed to contain at least 10 digits of precision, i.e., at least as much precision as a `double`, but probably more.
- Note that the `unsigned` modifier is not supported.
- C program snippet:

```
float z;
long double number;
```

THE UNIVERSITY of NEW MEXICO