Special Topics in Computer Science- CSC 4992

Number Systems

Numeric Types: `int`

- **`int`** is used for integers
- In many languages, the range of **`int`** is -2^{31} through $2^{31} - 1$ (-2,147,483,648 through 2,147,483,647)
- In Python, an integer's magnitude is limited only by the computer's memory

Computer Memory

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

…

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

Memory might have billions of *cells* that support the storage of trillions of binary digits or *bits* of information

Each cell in this memory has room for 32 bits

Bits and Bytes

$$\overbrace{0\;1\;0\;1\;1\;1\;1\;1}^{\text{byte}}\;\overbrace{0\;1\;0\;1\;1\;1\;1\;1}^{\text{byte}}\;\overbrace{0\;1\;0\;1\;1\;1\;1\;1}^{\text{byte}}\;\overbrace{0\;1\;0\;1\;1\;1\;1\;1}^{\text{byte}}$$

A byte is 8 bits

In some languages, **`int`** uses 4 bytes

The magnitude and the sign (+/-) of the number are determined by the binary representation

Decimal and Binary

- Decimal numbers use the 10 decimal digits and a base of 10
- Binary numbers use the binary digits 0 and 1 and a base of 2
- The base is often provided as a subscript to indicate the type of system, as in 3042_{10} and 11011110_2
- Thus, 1101_{10} represents a very different integer value from 1101_2

Positional Notation

- Number systems are *positional*, so the magnitude of the number depends on the base and the position of the digits in the number

- Each position represents a power of the number's base

- For example, in a 3-digit decimal number, the three positions represent the number of hundreds (10^2), tens (10^1), and ones (10^0)

- $342 = 3 * 10^2 + 4 * 10^1 + 2 * 10^0$
- $= 3 * 100 + 4 * 10 + 2 * 1$
- $= 300 + 40 + 2$
- $= 342$

Positional Notation: Binary

- The base is now 2 and the only digits are 0 and 1
- Each position represents a power of 2
- For example, in a 4-digit binary number, the four positions represent the number of eights (2^3), fours (2^2), twos (2^1), and ones (1^0)
- $1101 = 1 * 2^3 + 1 * 2^2 + 0 * 2^1 + 1 * 2^0$
- $= 1 * 8 + 1 * 4 + 0 * 2 + 1 * 1$
- $= 8 + 4 + 0 + 1$
- $= 13$

An Algorithm for Binary to Decimal Conversion

```
# Input: A string of 1 or more binary digits
# Output: The integer represented by the string
binary = input("Enter a binary number: ")
decimal = 0
exponent = len(binary) - 1
for digit in binary:
    decimal = decimal + int(digit) * 2 ** exponent
    exponent = exponent - 1
print("The integer value is", decimal)
```

The **`len`** function returns the number of characters in a string

The **`for`** loop visits each character in a string

Counting in Binary

Binary	Magnitude	
0	0	
1	1	
10	2	2^1
11	3	
100	4	2^2
101	5	
110	6	
111	7	
1000	8	2^3

Each power of 2 in binary is a 1 followed by the number of 0s equal to the exponent

Counting in Binary

Binary	Magnitude	
0	0	
1	1	$2^1 - 1$
10	2	
11	3	$2^2 - 1$
100	4	
101	5	
110	6	
111	7	$2^3 - 1$
1000	8	

Each number with only 1s equals one less than the power of 2 whose exponent is that number of 1s

Limits of Magnitude - Unsigned `int`s

- *Unsigned integers* are the non-negative integers
- The largest unsigned integer that can be represented using N bits is $2^N - 1$ (all bits are 1s)
- Thus, the largest unsigned integer stored in 32 bits is $2^{32} - 1$

Limits of Magnitude - Signed `int`s

- *Signed integers* include negative and positive integers and 0
- Part of the memory (one bit) must be reserved to represent the number’s sign somehow
- For each bit unavailable, you must subtract 1 from the exponent (2^{N-1}) of the number’s magnitude
- Thus, the largest positive signed integer stored in 32 bits is $2^{31} - 1$

Twos Complement Notation

- Positive numbers have 0 in the leftmost bit, negative numbers have 1 in the leftmost bit
- To compute a negative number's magnitude,
  - Invert all the bits
  - Add 1 to the result
  - Use the conversion algorithm
- To represent a negative number,
  - Translate the magnitude to an unsigned binary number
  - Invert all the bits
  - Add 1 to the result

Convert Decimal to Binary

- Start with an integer, N, and an empty string, S
- Assume that N > 0
- While N > 0:
  - Compute the remainder of dividing N by 2 (will be 0 or 1)
  - Prepend the remainder's digit to S
  - Reset N to the quotient of N and 2

An Algorithm for Decimal to Binary Conversion

```
# Input: An integer > 0
# Output: A string representing the integer in base 2
n = int(input("Enter an integer greater than 0: "))
binary = ''
while n > 0:
    rem = n % 2
    binary = str(rem) + binary
    n = n // 2
print(binary)
```

Here we want the quotient and the remainder, not exact division!