CS 460

Programming Languages

Fall 2021

Dr. Watts

(2 October 2023)

Project 1

- Framework now available

Data Types

- Collection of data values and a set of predefined operations on those values.
- User-defined - COBOL
- Abstract data types – Smalltalk - ALGOL
- Descriptor
  - Collection of the attributes of the variable
  - Amount and format of the memory associated with a variable

Primitive Data Types

- Numeric Types – binary representations
- Integer
  - Byte, short, int, long, long long
  - Unsigned vs. signed
  - Binary vs. twos compliment
- Floating Point
  - IEEE format
  - Sign bit, exponent, fraction
  - Precision vs. range – trade off
  - Non-terminating values (eg. 0.1)

Converting Binary to Decimal

- $1010011_2 == ?_{10}$

- 83 1*1 + 1*2 + 0*4 + 0*8 + 1*16 + 0*32 + 1*64 = 1+2+16+64 = 83
- Process? Repeated multiplication

Converting Decimal to Binary

- $326_{10} == ?_2$

- $326 = 256 + 64 + 4 + 2 == 101000110_2$?
- Process? Repeated division

What about arithmetic?

1010011_2 83

$+\ 1100110_2$ 102

10111001_2 => 185?

- 0+0 = 0
- 1+0 = 1
- 0+1 = 1
- 1+1 = 10
- 1+1+1 = 11

What about Negative Numbers?

- Sign bit

  11010011 => +83

  +01010011 => -83

  100100110 => +38 NOT 0!!!!!!!

  100100100?

  100000000?

  100100110?

Two’s Compliment

- 00000001 => 1 vs 11111111 => -1

  00101001 +83

  11010110 => One’s compliment

  \+ 1

  11010111 -83 => Two’s compliment

  00101001 +83

  1<u>00000000</u>

int storage?

- How many bits? => 32
- 1 bit for the sign
  - 1 => negative and 0 => positive
- 31 for the value
- 2^{31} patterns
- 000………0 => 0
- 100………0 => $-(2^{31})$
- $-(2^{31})$ => 0 => $2^{31}-1$ 0111…..1 + 1 = $-(2^{31})$
- for (int I = 1;I != 0; I++) cout << I << endl;

Other integer types

- short => 8 bits
- int => 16 bits
- long int => 32
- Now all int is 32
- 8 bit integer => signed char
- 8 bit unsigned => unsigned char
- 64 bits => long long
- unsigned int (UINT) 0 to 2^{32}-1

Converting Binary to Decimal

- $.1011_2 = ?_{10}$
  - $1 \text{ X } 0.5 + 0 \text{ x } 0.25 + 1 \text{ x } 0.125 + 1 \text{ x } 0.0625$
- Process?

Practice

- $0.1_2 = ?_{10}$
- $0.01_2 = ?_{10}$
- $0.011011_2 = ?_{10}$

Converting Decimal to Binary

- $.375_{10} = ?_2$ => .000375 vs .37500000
- Process?

Practice

- $.0625_{10} = ?_2$
- $0.1_{10} = ?_2$
- $0.01_{10} = ?_2$

IEEE Single Precision (float 32)

- The IEEE single precision floating point standard representation requires a 32 bit word, which may be represented as numbered from 0 to 31, left to right. The first bit is the sign bit, S, the next eight bits are the exponent bits, 'E', and the final 23 bits are the fraction 'F':

```
S EEEEEEEE FFFFFFFFFFFFFFFFFFFFFFF
0 1      8 9                    31
```

- The value V represented by the word may be determined as follows:
  - If E=255 and F is nonzero, then V=NaN ("Not a number")
  - If E=255 and F is zero and S is 1, then V=-Infinity
  - If E=255 and F is zero and S is 0, then V=Infinity
  - If 0<E<255 then V=(-1)**S * 2 ** (E-127) * (1.F) where "1.F" is intended to represent the binary number created by prefixing F with an implicit leading 1 and a binary point.
  - If E=0 and F is nonzero, then V=(-1)**S * 2 ** (-126) * (0.F) These are "unnormalized" values.
  - If E=0 and F is zero and S is 1, then V=-0
  - If E=0 and F is zero and S is 0, then V=0
- In particular,
  - `0 00000000 00000000000000000000000 = 0`
  - `1 00000000 00000000000000000000000 = -0`
  - `0 11111111 00000000000000000000000 = Infinity`
  - `1 11111111 00000000000000000000000 = -Infinity`
  - `0 11111111 00000100000000000000000 = NaN`
  - `1 11111111 00100010001001010101010 = NaN`
  - `0 10000000 00000000000000000000000 = +1 * 2**(128-127) * 1.0 = 2`
  - `0 10000001 10100000000000000000000 = +1 * 2**(129-127) * 1.101 = 6.5`
  - `1 10000001 10100000000000000000000 = -1 * 2**(129-127) * 1.101 = -6.5`
  - `0 00000001 00000000000000000000000 = +1 * 2**(1-127) * 1.0 = 2**(-126)`
  - `0 00000000 10000000000000000000000 = +1 * 2**(-126) * 0.1 = 2**(-127)`
  - `0 00000000 00000000000000000000001 = +1 * 2**(-126) * 0.00000000000000000000001 = 2**(-149) (Smallest positive value)`

IEEE Double Precision (float64)

- The IEEE double precision floating point standard representation requires a 64 bit word, which may be represented as numbered from 0 to 63, left to right. The first bit is the sign bit, S, the next eleven bits are the exponent bits, 'E', and the final 52 bits are the fraction 'F':

```
S EEEEEEEEEEE FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
0 1        11 12                                                63
```

- The value V represented by the word may be determined as follows:
- If E=2047 and F is nonzero, then V=NaN ("Not a number")
- If E=2047 and F is zero and S is 1, then V=-Infinity
- If E=2047 and F is zero and S is 0, then V=Infinity
- If 0<E<2047 then V=(-1)**S * 2 ** (E-1023) * (1.F) where "1.F" is intended to represent the binary number created by prefixing F with an implicit leading 1 and a binary point.
- If E=0 and F is nonzero, then V=(-1)**S * 2 ** (-1022) * (0.F) These are "unnormalized" values.
- If E=0 and F is zero and S is 1, then V=-0
- If E=0 and F is zero and S is 0, then V=0

IEEE Quadruple Precision (float128)

- 1 bit sign
- 15 bit exponent
- 112 bit fraction

Half Precision (float16)

- Half precision floating point representation requires a 16 bit word, which may be represented as numbered from 0 to 15, left to right. The first bit is the sign bit, S, the next five bits are the exponent bits, 'E', and the final 10 bits are the fraction 'F':

```
S EEEEE FFFFFFFFFF
0 1   6 6        15
```

- The value V represented by the word may be determined as follows
- Sign
  - Sign = 0 is positive
  - Sign = 1 is negative
- Exponent
  - Biased
  - 00000 – 11111
- Fraction

Half Precision

- Example: 125.25
  - Whole number $125_{10} == ?_2$
  - Fraction $.25_{10} == ?_2$

Position	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Bit value																