CS 460

Programming Languages

Fall 2021

Dr. Watts

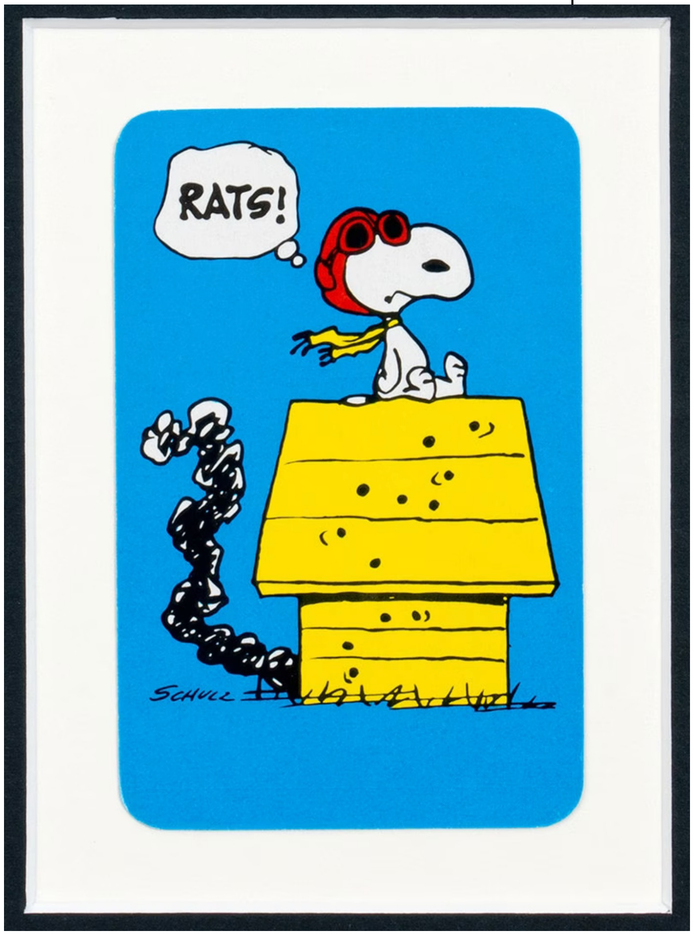
(4 October 2023)

Project 1

- Questions?

Exercise 2

- Considering revisiting this as Exercise 5
- Issues
  - Testing
  - C++ input protocols
  - Modifying public section of class
  - Following the spec
  - RATS! : Read All The Spec!



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

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 X 0.5 + 0 x 0.25 + 1 x 0.125 + 1 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   5 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 1

- 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																

Half Precision Example 2

- Example: 123000.0
  - Whole number $123000_{10} == ?_2$
  - Fraction $.0_{10} == ?_2$

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

Half Precision Example 3

- Example: .1259765625
  - Whole number $0_{10} == ?_2$
  - Fraction $.1259765625_{10} == ?_2$

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

Half Precision Example 4

- Example: 25.20
  - Whole number $25_{10} == ?_2$
  - Fraction $.20_{10} == ?_2$

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

Half Precision Example 5

- Example: 125.20
  - Whole number $125_{10} == ?_2$
  - Fraction $.20_{10} == ?_2$

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

Let's go the other way!

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

Let’s go the other way!

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

Some Half-precision values

Binary	Hex	Value	Notes
0 00000 0000000000	0000	0	
0 00000 0000000001	0001	$2^{-14} \times (0 + \frac{1}{1024}) \approx 0.000000059604645$	smallest positive subnormal number
0 00000 1111111111	03ff	$2^{-14} \times (0 + \frac{1023}{1024}) \approx 0.000060975552$	largest subnormal number
0 00001 0000000000	0400	$2^{-14} \times (1 + \frac{0}{1024}) \approx 0.00006103515625$	smallest positive normal number
0 01101 0101010101	3555	$2^{-2} \times (1 + \frac{341}{1024}) \approx 0.33325195$	nearest value to 1/3
0 01110 1111111111	3bff	$2^{-1} \times (1 + \frac{1023}{1024}) \approx 0.99951172$	largest number less than one
0 01111 0000000000	3c00	$2^{0} \times (1 + \frac{0}{1024}) = 1$	one
0 01111 0000000001	3c01	$2^{0} \times (1 + \frac{1}{1024}) \approx 1.00097656$	smallest number larger than one
0 11110 1111111111	7bff	$2^{15} \times (1 + \frac{1023}{1024}) = 65504$	largest normal number
0 11111 0000000000	7c00	∞	infinity
1 00000 0000000000	8000	-0	
1 10000 0000000000	c000	-2	
1 11111 0000000000	fc00	$-\infty$	negative infinity

```
#include <iostream>
#include <iomanip>
#include <cmath>
using namespace std;

int main ()
{
        float value = 0;
        do
        {
                cout << "Please enter a floating point value (0 to quit): ";
                cin >> value;
                cout << "  value: " << fixed << showpoint << setprecision (30)
                     << value << endl;
                float rounded = (round (value * 100)) / 100.0;
                cout << "rounded: " << fixed << showpoint << setprecision (30)
                     << rounded << endl;
        } while (value != 0);
        return 0;
}
```

Other Numeric Types

- Arbitrary precision
- Fraction type
- User defined types

String types

- C style strings
- C++ strings

Composite Data Types

- Array
- Struct
- Class
- Union