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 l = 1; l != 0; l++) cout << l << 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

- \(.1011_2 = ?_{10}\)
  - \(1 \times 0.5 + 0 \times 0.25 + 1 \times 0.125 + 1 \times 0.0625\)

- Process?
Practice

- $0.1_2 = ?_{10}$
- $0.01_2 = ?_{10}$
- $0.011011_2 = ?_{10}$
Converting Decimal to Binary

- \(0.375_{10} = ?_2 \Rightarrow 0.000375 \text{ vs } 0.37500000\)
- Process?
Practice

- \(0.0625_{10} = ?_2\)
- \(0.1_{10} = ?_2\)
- \(0.01_{10} = ?_2\)
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 FFFFFFFFFFFFFFFFFFFFFFFF
  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**(128)
  - 0 00000000 10000000000000000000000 = +1 * 2**((-126) * 0.1) = 2**((-127))
  - 0 00000000 00000000000000000000001 = +1 * 2**((-126) * 0.000000000000000000000001)
  = 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
```

- 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
```

- 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 $125_{10} \equiv ?_2$
  - Fraction $.25_{10} \equiv ?_2$

<table>
<thead>
<tr>
<th>Position</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit value</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Half Precision Example 2

- Example: 123000.0
  - Whole number $123000_{10} == ?_2$
  - Fraction $.0_{10} == ?_2$
Half Precision Example 3

- Example: \(0.1259765625\)
  - Whole number \(0_{10} \equiv ?_2\)
  - Fraction \(0.1259765625_{10} \equiv ?_2\)
Half Precision Example 4

- Example: 25.20
  - Whole number $25_{10} = ?_2$
  - Fraction $0.20_{10} = ?_2$
Half Precision Example 5

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

<table>
<thead>
<tr>
<th>Position</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit value</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Let’s go the other way!

<table>
<thead>
<tr>
<th>Position</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit value</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
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

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