IEEE Floating-Point Representation\textsuperscript{1}

$x = (-1)^s \times M \times 2^E$

- The sign $s$ determines whether the number is negative ($s = 1$) or positive ($s = 0$).
- The significand $M$ is a fractional binary number that ranges either between $1$ and $2 - \epsilon$, or between $0$ and $1 - \epsilon$.
- The exponent $E$ weights the value by a (possibly negative) power of $2$.

\textsuperscript{1}William Kahan (1985); Aka IEEE754.
• That is why we call a **double** value.

---

\(^2\)See Figure 2-31 in Byrant, p. 104.
Assignments

• An assignment statement designates a value to the variable.

```java
int x; // make a variable declaration
...
x = 1; // assign 1 to x
```

• The equal sign (=) is used as the assignment operator.
  • For example, is the expression \( x = x + 1 \) correct?
  • Direction: from the right-hand side to the left-hand side
• To assign a value to a variable, you must place the variable name to the left of the assignment operator.\(^3\)
  • For example, \( 1 = x \) is wrong.
  • \( 1 \) cannot be resolved to a memory space.

\(^3\)\( x \) can be a l-value and r-value, but \( 1 \) and other numbers can be only r-value but not l-value. See Value.
Two “Before” Rules

- Every variable has a **scope**.
  - The scope of a variable is the range of the program where the variable can be referenced.\(^4\)
- A variable must be declared before it can be assigned a value.
  - In practice, do not declare the variable until you need it.
- A declared variable must be assigned a value before it can be used.\(^5\)

---

\(^4\) The detail of variable scope is introduced later.

\(^5\) In symbolic programming, such as Mathematica and Maple, a variable can be manipulated without assigning a value. For example, \(x + x\) returns \(2x\).
## Arithmetic Operators

### Table 2-3 in YDL, p. 46.

<table>
<thead>
<tr>
<th>Name</th>
<th>Meaning</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
<td>34 + 1</td>
<td>35</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
<td>34.0 – 0.1</td>
<td>33.9</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
<td>300 * 30</td>
<td>9000</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
<td>1.0 / 2.0</td>
<td>0.5</td>
</tr>
<tr>
<td>%</td>
<td>Remainder</td>
<td>20 % 3</td>
<td>2</td>
</tr>
</tbody>
</table>

- Note that the operator depends on the operands involved.

---

[6] See Table 2-3 in YDL, p. 46.
Tricky Pitfalls

• Can you explain this result?

```java
... double x = 1 / 2;
 System.out.println(x); // output?
...
```

• Revisit \(0.5 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1 = 0.7\)

```java
... System.out.println(1 / 2 - 1 / 10 - 1 / 10 - 1 / 10 - 1 / 10); // output 0; however, this is not the real solution to the original problem.
...
```

\(^7\)Thanks to a lively discussion on on June 7, 2016.
Type Conversion and Compatibility

- If a type is **compatible** to another, then the compiler will perform the conversion **implicitly**.
  - For example, the integer 1 is compatible to a double value 1.0.
- However, there is no automatic conversion from double to int.
  (Why?)
- To do so, you must use a **cast**, which performs an explicit conversion for compilation.
- Similarly, a long value is not compatible to int.
Casting

```java
int x = 1;
double y = x; // compatible; implicit conversion
x = y; // incompatible; need an explicit conversion by casting
x = (int) y; // succeed!!
```

- Note that the Java compiler does only **type-checking** but no real execution before compilation.
- In other words, the values of `x` and `y` are unknown until they are really executed.
Type Conversion and Compatibility (concluded)

- small-size types $\rightarrow$ large-size types
- small-size types $\leftrightarrow$ large-size types (need a cast)
- simple types $\rightarrow$ complicated types
- simple types $\leftrightarrow$ complicated types (need a cast)
Characters

- A character stored by the machine is represented by a sequence of 0’s and 1’s.
  - For example, ASCII code. (See the next page.)
- The `char` type is a 16-bit unsigned primitive data type.\(^8\)

\(^8\)Java uses **Unicode** to represent characters. Unicode defines a fully international character set that can represent all of the characters found in all human languages.
<table>
<thead>
<tr>
<th>Hex</th>
<th>Dec Char</th>
<th>Hex</th>
<th>Dec Char</th>
<th>Hex</th>
<th>Dec Char</th>
<th>Hex</th>
<th>Dec Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>0</td>
<td>0x20</td>
<td>Space</td>
<td>0x40</td>
<td>@</td>
<td>0x60</td>
<td></td>
</tr>
<tr>
<td>0x01</td>
<td>SOH</td>
<td>0x21</td>
<td>!</td>
<td>0x41</td>
<td>A</td>
<td>0x61</td>
<td></td>
</tr>
<tr>
<td>0x02</td>
<td>STX</td>
<td>0x22</td>
<td>&quot;</td>
<td>0x42</td>
<td>B</td>
<td>0x62</td>
<td></td>
</tr>
<tr>
<td>0x03</td>
<td>ETX</td>
<td>0x23</td>
<td>#</td>
<td>0x43</td>
<td>C</td>
<td>0x63</td>
<td></td>
</tr>
<tr>
<td>0x04</td>
<td>EOT</td>
<td>0x24</td>
<td>$</td>
<td>0x44</td>
<td>D</td>
<td>0x64</td>
<td></td>
</tr>
<tr>
<td>0x05</td>
<td>ENQ</td>
<td>0x25</td>
<td>%</td>
<td>0x45</td>
<td>E</td>
<td>0x65</td>
<td></td>
</tr>
<tr>
<td>0x06</td>
<td>ACK</td>
<td>0x26</td>
<td>&amp;</td>
<td>0x46</td>
<td>F</td>
<td>0x66</td>
<td></td>
</tr>
<tr>
<td>0x07</td>
<td>BELL</td>
<td>0x27</td>
<td>'</td>
<td>0x47</td>
<td>G</td>
<td>0x67</td>
<td></td>
</tr>
<tr>
<td>0x08</td>
<td>BS</td>
<td>0x28</td>
<td>(</td>
<td>0x48</td>
<td>H</td>
<td>0x68</td>
<td></td>
</tr>
<tr>
<td>0x09</td>
<td>TAB</td>
<td>0x29</td>
<td>)</td>
<td>0x49</td>
<td>I</td>
<td>0x69</td>
<td></td>
</tr>
<tr>
<td>0x0A</td>
<td>LF</td>
<td>0x2A</td>
<td>*</td>
<td>0x4A</td>
<td>J</td>
<td>0x6A</td>
<td></td>
</tr>
<tr>
<td>0x0B</td>
<td>VT</td>
<td>0x2B</td>
<td>+</td>
<td>0x4B</td>
<td>K</td>
<td>0x6B</td>
<td></td>
</tr>
<tr>
<td>0x0C</td>
<td>FF</td>
<td>0x2C</td>
<td>,</td>
<td>0x4C</td>
<td>L</td>
<td>0x6C</td>
<td></td>
</tr>
<tr>
<td>0x0D</td>
<td>CR</td>
<td>0x2D</td>
<td>-</td>
<td>0x4D</td>
<td>M</td>
<td>0x6D</td>
<td></td>
</tr>
<tr>
<td>0x0E</td>
<td>SO</td>
<td>0x2E</td>
<td>.</td>
<td>0x4E</td>
<td>N</td>
<td>0x6E</td>
<td></td>
</tr>
<tr>
<td>0x0F</td>
<td>SI</td>
<td>0x2F</td>
<td>/</td>
<td>0x4F</td>
<td>O</td>
<td>0x6F</td>
<td></td>
</tr>
<tr>
<td>0x10</td>
<td>DLE</td>
<td>0x30</td>
<td>0</td>
<td>0x50</td>
<td>P</td>
<td>0x70</td>
<td></td>
</tr>
<tr>
<td>0x11</td>
<td>DC1</td>
<td>0x31</td>
<td>1</td>
<td>0x51</td>
<td>Q</td>
<td>0x71</td>
<td></td>
</tr>
<tr>
<td>0x12</td>
<td>DC2</td>
<td>0x32</td>
<td>2</td>
<td>0x52</td>
<td>R</td>
<td>0x72</td>
<td></td>
</tr>
<tr>
<td>0x13</td>
<td>DC3</td>
<td>0x33</td>
<td>3</td>
<td>0x53</td>
<td>S</td>
<td>0x73</td>
<td></td>
</tr>
<tr>
<td>0x14</td>
<td>DC4</td>
<td>0x34</td>
<td>4</td>
<td>0x54</td>
<td>T</td>
<td>0x74</td>
<td></td>
</tr>
<tr>
<td>0x15</td>
<td>NAK</td>
<td>0x35</td>
<td>5</td>
<td>0x55</td>
<td>U</td>
<td>0x75</td>
<td></td>
</tr>
<tr>
<td>0x16</td>
<td>SYN</td>
<td>0x36</td>
<td>6</td>
<td>0x56</td>
<td>V</td>
<td>0x76</td>
<td></td>
</tr>
<tr>
<td>0x17</td>
<td>ETB</td>
<td>0x37</td>
<td>7</td>
<td>0x57</td>
<td>W</td>
<td>0x77</td>
<td></td>
</tr>
<tr>
<td>0x18</td>
<td>CAN</td>
<td>0x38</td>
<td>8</td>
<td>0x58</td>
<td>X</td>
<td>0x78</td>
<td></td>
</tr>
<tr>
<td>0x19</td>
<td>EM</td>
<td>0x39</td>
<td>9</td>
<td>0x59</td>
<td>Y</td>
<td>0x79</td>
<td></td>
</tr>
<tr>
<td>0x1A</td>
<td>SUB</td>
<td>0x3A</td>
<td>:</td>
<td>0x5A</td>
<td>Z</td>
<td>0x7A</td>
<td></td>
</tr>
<tr>
<td>0x1B</td>
<td>FSC</td>
<td>0x3B</td>
<td>;</td>
<td>0x5B</td>
<td>[</td>
<td>0x7B</td>
<td></td>
</tr>
<tr>
<td>0x1C</td>
<td>FS</td>
<td>0x3C</td>
<td>&lt;</td>
<td>0x5C</td>
<td>\</td>
<td>0x7C</td>
<td></td>
</tr>
<tr>
<td>0x1D</td>
<td>GS</td>
<td>0x3D</td>
<td>=</td>
<td>0x5D</td>
<td>]</td>
<td>0x7D</td>
<td></td>
</tr>
<tr>
<td>0x1E</td>
<td>RS</td>
<td>0x3E</td>
<td>&gt;</td>
<td>0x5E</td>
<td>^</td>
<td>0x7E</td>
<td></td>
</tr>
<tr>
<td>0x1F</td>
<td>US</td>
<td>0x3F</td>
<td>?</td>
<td>0x5F</td>
<td>_</td>
<td>0x7F</td>
<td></td>
</tr>
</tbody>
</table>
Example

- Characters can also be used as (positive) integers on which you can perform arithmetic operations.\(^9\)
- For example,

```java
... 
char x = 'a'; // single-quoted: a char value
System.out.println(x + 1); // output 98!!
System.out.println((char)(x + 1)); // output b

String s = "Java"; // double-quoted: a String object

... 
```

- You can imagine that a String object comprises characters equipped with plentiful tools.\(^{10}\)

---


\(^{10}\)As an analogy, a molecule (string) consists of atoms (characters).
Boolean Values

• The program is supposed to do decision making by itself, for example, Google Driverless Car.\textsuperscript{11}

• To do this, Java has the boolean-type flow controls (selections and iterations).

• This type has only two possible values, true and false.

• Note that a boolean value cannot be cast into a value of another type, nor can a value of another type be cast into a boolean value. (Why?)

\textsuperscript{11}\textsuperscript{11}See https://www.google.com/selfdrivingcar/.
Rational Operators

<table>
<thead>
<tr>
<th>Java Operator</th>
<th>Mathematics Symbol</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>&lt;</td>
<td>less than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>≤</td>
<td>less than or equal to</td>
</tr>
<tr>
<td>&gt;</td>
<td>&gt;</td>
<td>greater than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>≥</td>
<td>greater than or equal to</td>
</tr>
<tr>
<td>==</td>
<td>=</td>
<td>equal to</td>
</tr>
<tr>
<td>!=</td>
<td>≠</td>
<td>not equal to</td>
</tr>
</tbody>
</table>

- These operators take two operands.
- Rational expressions return a boolean value.
- Note that the equality operator is double equality sign (==), not single equality sign (=).

\(^{12}\)See Table 3-1 in YDL, p. 82.
Example

\begin{verbatim}
... int x = 2;
boolean a = x > 1;
boolean b = x < 1;
boolean c = x == 1;
boolean d = x != 1;
boolean e = 1 < x < 3; // sorry?
...
\end{verbatim}

- Be aware that e is logically correct but \textit{syntactically wrong}.
- Usually, the boolean expression consists of a combination of rational expressions.
  - For example, $1 < x < 3$ should be $(1 < x) \&\& (x < 3)$, where \&\& refers to the AND operator.
# Logical Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>not</td>
<td>logical negation</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>and</td>
<td>logical conjunction</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>^</td>
<td>exclusive or</td>
<td>logical exclusion</td>
</tr>
</tbody>
</table>

> See Table 3-2 in YDL, p. 102.
Truth Table

- Let $X$ and $Y$ be two Boolean variables.
- Then the truth table for logical operators is as follows:

| $X$ | $Y$ | $!X$ | $X$ && $Y$ | $X$ || $Y$ | $X$ ^ $Y$ |
|-----|-----|------|------------|---------|-----------|
| T   | T   | F    | T          | T       | T         |
| T   | F   | F    | F          | T       | T         |
| F   | T   | T    | F          | T       | T         |
| F   | F   | T    | F          | F       | F         |

- Note that the instructions of computers, such as arithmetic operations, are implemented by logic gates.\(^{14}\)

\(^{14}\)See any textbook for digital circuit design.
“Logic is the anatomy of thought.”

– John Locke (1632–1704)

“This sentence is false.”

– anonymous

“I know that I know nothing.”

– Plato

(In Apology, Plato relates that Socrates accounts for his seeming wiser than any other person because he does not imagine that he knows what he does not know.)
## Arithmetic Compound Assignment Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>++</td>
<td>Increment</td>
</tr>
<tr>
<td>+=</td>
<td>Addition assignment</td>
</tr>
<tr>
<td>-=</td>
<td>Subtraction assignment</td>
</tr>
<tr>
<td>*=</td>
<td>Multiplication assignment</td>
</tr>
<tr>
<td>/=</td>
<td>Division assignment</td>
</tr>
<tr>
<td>%=</td>
<td>Modulus assignment</td>
</tr>
<tr>
<td>--</td>
<td>Decrement</td>
</tr>
</tbody>
</table>

- Note that these shorthand operators are not available in languages such as Matlab and R.
Example

```java
int x = 1;
System.out.println(x); // output 1
x = x + 1;
System.out.println(x); // output 2
x += 2;
System.out.println(x); // output 4
x++; // equivalent to x += 1 and x = x + 1
System.out.println(x); // output 5
```

...
• The compound assignment operators are also useful for char values. \(^{15}\)

• For example,

```java
... 
char s = 'a';
System.out.println(s); // output a
s += 1;
System.out.println(s); // output b
s++; 
System.out.println(s); // output c
... 
```

\(^{15}\)Contribution by Mr. Edward Wang (Java265) on May 1, 2016.
The expression `++x` first increments the value of `x` and then returns `x`.

Instead, the expression `x++` first returns the value of `x` and then increments itself.

For example,

```java
... 
int x = 1;
int y = ++x;
System.out.println(y); // output 2; aka preincrement
System.out.println(x); // output 2

int w = 1;
int z = w++;
System.out.println(z); // output 1; aka postincrement
System.out.println(w); // output 2

... 
```

We will use these notations very often.
## Operator Precedence

<table>
<thead>
<tr>
<th>Precedence</th>
<th>Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>var++</code> and <code>var--</code> (Postfix)</td>
</tr>
<tr>
<td></td>
<td><code>+</code>, <code>-</code> (Unary plus and minus), <code>++var</code> and <code>--var</code> (Prefix)</td>
</tr>
<tr>
<td></td>
<td>(type) (Casting)</td>
</tr>
<tr>
<td></td>
<td><code>!</code> (Not)</td>
</tr>
<tr>
<td></td>
<td><code>*</code>, <code>/</code>, <code>%</code> (Multiplication, division, and remainder)</td>
</tr>
<tr>
<td></td>
<td><code>+</code>, <code>-</code> (Binary addition and subtraction)</td>
</tr>
<tr>
<td></td>
<td><code>&lt;</code>, <code>&lt;=</code>, <code>&gt;</code>, <code>&gt;=</code> (Comparison)</td>
</tr>
<tr>
<td></td>
<td><code>==</code>, <code>!=</code> (Equality)</td>
</tr>
<tr>
<td></td>
<td><code>^</code> (Exclusive OR)</td>
</tr>
<tr>
<td></td>
<td><code>&amp;&amp;</code> (AND)</td>
</tr>
<tr>
<td></td>
<td>`</td>
</tr>
<tr>
<td></td>
<td><code>=</code>, <code>+=</code>, <code>-=</code>, <code>*=</code>) <code>/=</code>, <code>%=</code> (Assignment operator)</td>
</tr>
</tbody>
</table>

---

\[16\] See Table 3-10 in YDL, p. 116.
Using Parentheses

- Parentheses are used in expressions to change the natural order of precedence among the operators.
- One always evaluates the expression inside of parentheses first.
Scanner Objects

- It is not convenient to modify the source code and recompile it for a different radius.
- Reading from the console enables the program to receive an input from the user.
- A `Scanner` object provides some input methods, say the input received from the keyboard or the files.
- Java uses `System.in` to refer to the standard input device, by default, the keyboard.
Write a program which receives a number as input, and outputs the area of the circle.

```java
import java.util.Scanner;
...
Scanner input = new Scanner(System.in);
System.out.println("Enter r?");
// input
int r = input.nextInt();
// algorithm
double area = r * r * 3.14;
// output
System.out.println(area);
input.close();
...
- In the listing, Line 3 is to create a **Scanner** object by the `new` operator, as an agent between the keyboard and your program.
- Note that all objects are resided in the **heap** of the memory.
- To control this object, its **memory address** is then assigned to the variable `input` which is a variable in the **stack** of memory.
- So the variable `input` is a **reference**.
- We will discuss the objects and reference variables later.
Methods Provided by Scanner Objects

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>nextByte()</code></td>
<td>reads an integer of the <code>byte</code> type.</td>
</tr>
<tr>
<td><code>nextShort()</code></td>
<td>reads an integer of the <code>short</code> type.</td>
</tr>
<tr>
<td><code>nextInt()</code></td>
<td>reads an integer of the <code>int</code> type.</td>
</tr>
<tr>
<td><code>nextLong()</code></td>
<td>reads an integer of the <code>long</code> type.</td>
</tr>
<tr>
<td><code>nextFloat()</code></td>
<td>reads a number of the <code>float</code> type.</td>
</tr>
<tr>
<td><code>nextDouble()</code></td>
<td>reads a number of the <code>double</code> type.</td>
</tr>
<tr>
<td><code>next()</code></td>
<td>reads a string that ends before a whitespace character.</td>
</tr>
<tr>
<td><code>nextLine()</code></td>
<td>reads a line of text (i.e., a string ending with the <code>Enter</code> key pressed).</td>
</tr>
</tbody>
</table>

\(^{17}\)See Table 2-1 in YDL, p. 38.
Example: Mean and Standard Deviation

Write a program which calculates the mean and the standard deviation of 3 numbers.

• The mean of 3 numbers is given by \( \bar{x} = \left( \sum_{i=1}^{3} x_i \right) / 3 \).
• Also, the resulting standard deviation is given by

\[
S = \sqrt{\frac{\sum_{i=1}^{3} (x_i - \bar{x})^2}{3}}.
\]

• You may use these two methods:
  • Math.pow(double x, double y) for \( x^y \)
  • Math.sqrt(double x) for \( \sqrt{x} \)
• See more methods within Math class.
Scanner input = new Scanner(System.in);
System.out.println("a = ?");
double a = input.nextDouble();
System.out.println("b = ?");
double b = input.nextDouble();
System.out.println("c = ?");
double c = input.nextDouble();

double mean = (a + b + c) / 3;
double std = Math.sqrt((Math.pow(a - mean, 2) +
                        Math.pow(b - mean, 2) +
                        Math.pow(c - mean, 2)) / 3);

System.out.println("mean = " + mean);
System.out.println("std = " + std);
class Lecture3 {
    "Selections"
}

// Keywords
if, else, else if, switch, case, default
Flow Controls

The basic algorithm (and program) is constituted by the following operations:

- **Sequential statements**: execute instructions in order.
- **Selection**: first check if the predetermined condition is satisfied, then execute the corresponding instruction.
- **Repetition**: repeat the execution of some instructions until the criterion fails.
• Note that they are involved with each other generally.
• For example, recall how to find the maximum in the input list?
Selections

- One-way if statements
- Two-way if-else statements
- Nested if statements
- Multiway if-else if-else statements
- switch-case statements
- Conditional operators
A one-way if statement executes an action if and only if the condition is true.
The keyword `if` is followed by the parenthesized condition. The condition should be a boolean expression or a boolean value. If the condition is true, then the statements in the selection body will be executed once. If not, then the program won’t enter the selection body and skip the whole selection body. Note that the braces can be omitted if the block contains only single statement.
Write a program which receives a nonnegative number as input for the radius of a circle, and determines the area of the circle.

```java
... 

    double area;
    if (r > 0) {
        area = r * r * 3.14;
        System.out.println(area);
    }

... 
```

- However, the world is not well-defined.
Two-Way if-else Statements

A two-way if-else statement decides which statements to execute based on whether the condition is true or false.

```java
... if (condition) {
    // body for the true case
} else {
    // body for the false case
}
...
```
Flowchart:
- **boolean-expression**
  - true
    - Statement(s) for the true case
  - false
    - Statement(s) for the false case
Write a program which receives a number as input for the radius of a circle. If the number is nonnegative, then determine the area of the circle; otherwise, output “Not a circle.”

```java
... double area;
if (r > 0) {
    area = r * r * 3.14;
    System.out.println(area);
} else {
    System.out.println("Not a circle.");
}
input.close();
...
Nested if Statements

- For example,

```java
... if (score >= 90) 
    System.out.println("A");
else {
    if (score >= 80) 
        System.out.println("B");
    else {
        if (score >= 70) 
            System.out.println("C");
        else {
            if (score >= 60) 
                System.out.println("D");
            else 
                System.out.println("F");
        }
    }
}
... 
```
Multi-Way if-else

• Let’s redo the previous problem.

```java
... if (score >= 90) 
    System.out.println("A");
else if (score >= 80)
    System.out.println("B");
else if (score >= 70)
    System.out.println("C");
else if (score >= 60)
    System.out.println("D");
else
    System.out.println("F");
...
```

• An if-elseif-else statement is a preferred format for multiple alternatives, in order to avoid deep indentation and make the program easy to read.
The order of conditions may be relevant. (Why?)

```java
... if (score >= 90 && score <= 100)
else if (score >= 80 && score < 90)
... else ...
```

The performance may degrade due to the order of conditions. (Why?)
Common Errors

```java
... double area;
if (r > 0);
    area = r * r * 3.14;
System.out.println(area);
...
```