3: Math
Outline

❖ Operations

❖ Expressions

❖ More Arithmetic Operations

❖ Logical Operations
Operations

❖ *Arithmetic*

+ - * / %  
&& || !
Arithmetic Operations

Four basic: addition $+$, subtraction $-$, multiplication $\ast$, division $/$

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1+2</td>
<td>3</td>
</tr>
<tr>
<td>1-2</td>
<td>-1</td>
</tr>
<tr>
<td>2*(-3)</td>
<td>-6</td>
</tr>
<tr>
<td>1.0/2.0</td>
<td>0.5</td>
</tr>
<tr>
<td>1.0/0.0</td>
<td>inf</td>
</tr>
<tr>
<td>-1.0/0.0</td>
<td>-inf</td>
</tr>
<tr>
<td>0.0/0.0</td>
<td>nan</td>
</tr>
<tr>
<td>1/2</td>
<td>0</td>
</tr>
</tbody>
</table>

$+$,$-\ast$ of all numbers, and $/$ of reals work as expected

$/$ of integers yields an integer - needs definition
Integer Division

- \( a, b - \text{integers}, \frac{a}{b} \) is also an \( \text{integer} \)
- **Round toward 0** (truncation) of actual ratio

<table>
<thead>
<tr>
<th>fraction</th>
<th>actual value</th>
<th>after truncation</th>
</tr>
</thead>
<tbody>
<tr>
<td>3/2</td>
<td>1 1/2</td>
<td>1</td>
</tr>
<tr>
<td>8/3</td>
<td>2 2/3</td>
<td>2</td>
</tr>
<tr>
<td>15/4</td>
<td>3 3/4</td>
<td>3</td>
</tr>
<tr>
<td>4/5</td>
<td>4/5</td>
<td>0</td>
</tr>
</tbody>
</table>

Positive

<table>
<thead>
<tr>
<th>fraction</th>
<th>actual value</th>
<th>after truncation</th>
</tr>
</thead>
<tbody>
<tr>
<td>3/-2</td>
<td>-1 1/2</td>
<td>-1</td>
</tr>
<tr>
<td>-8/3</td>
<td>-2 2/3</td>
<td>-2</td>
</tr>
<tr>
<td>-15/-4</td>
<td>3 3/4</td>
<td>3</td>
</tr>
<tr>
<td>-4/5</td>
<td>-4/5</td>
<td>0</td>
</tr>
</tbody>
</table>

Negative

- **Division by 0** - error
- as + but with appropriate sign
Modulo

- $a, b$ - integers, $a$ **modulo** (or **mod**)$b$, written $a \% b$, is the **remainder** when $a$ is divided by $b$.

Divide $a$ marbles equally among $b$ kids
Each gets: $\frac{a}{b}$
You have left: $a \% b$

- **What if $a$ or $b$ is negative?**

- **Always:** $\left(\frac{a}{b}\right) \times b + a \% b = a$

<table>
<thead>
<tr>
<th>$a$</th>
<th>$b$</th>
<th>$a/b$</th>
<th>$(a/b) \times b$</th>
<th>$a % b$</th>
<th>$(a/b) \times b + a % b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>4</td>
<td>3</td>
<td>12</td>
<td>1</td>
<td>13</td>
</tr>
<tr>
<td>13</td>
<td>-4</td>
<td>-3</td>
<td>12</td>
<td>1</td>
<td>13</td>
</tr>
<tr>
<td>-13</td>
<td>4</td>
<td>-3</td>
<td>-12</td>
<td>-1</td>
<td>-13</td>
</tr>
<tr>
<td>-13</td>
<td>-4</td>
<td>3</td>
<td>-12</td>
<td>-1</td>
<td>-13</td>
</tr>
</tbody>
</table>

- **Mod 0 - error**

 Operations.c
Outline

❖ Operations

❖ Expressions

❖ More Arithmetic Operations

❖ Logical Operations
Outline

❖ Operations

❖ Expressions

❖ More Arithmetic Operations

❖ Logical Operations
Expressions

❖ Constant, variable, or their valid **combination using operators**

❖ Every expression has a **value** and a **type**

\[ \text{Valid: } 'a', 77, x, 3 + y \mod z \]

\[ \text{Invalid: } 77', 3a, \text{for}, 3 \div y \]
Determined by the operations, performed sequentially
Need to define operation order

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1+2*3</td>
<td>7</td>
<td>Multiplication <em>precedes</em> addition</td>
</tr>
<tr>
<td>(1+2)*3</td>
<td>9</td>
<td>Parentheses alter evaluation order</td>
</tr>
<tr>
<td>1-2+3</td>
<td>2</td>
<td>Additions/subtractions evaluated <em>left to right</em></td>
</tr>
<tr>
<td>1-(2+3)</td>
<td>-4</td>
<td>Parentheses alter evaluation order</td>
</tr>
<tr>
<td>1.0/2.0*3.0</td>
<td>1.5</td>
<td>Multiplications/divisions evaluated <em>left to right</em></td>
</tr>
<tr>
<td>1/2*3</td>
<td>0</td>
<td>1/2 evaluated to 0</td>
</tr>
<tr>
<td>4%2*3</td>
<td>0</td>
<td>Same priority, evaluated <em>left to right</em></td>
</tr>
</tbody>
</table>
Type

❖ Arithmetic operations on same type operands result in same type
❖ Arithmetic operations on different types convert, or promote, operands to the higher type according to

\[
\text{char} \rightarrow \text{short} \rightarrow \text{int} \rightarrow \text{long} \rightarrow \text{float} \rightarrow \text{double}
\]

\[
2 / 4.0 \rightarrow 2.0 / 4.0 \rightarrow 0.5
\]

❖ Only exception to above rules is that result is always at least int
### Examples

#### Declarations

<table>
<thead>
<tr>
<th>Declaration</th>
</tr>
</thead>
<tbody>
<tr>
<td>char c;</td>
</tr>
<tr>
<td>double d;</td>
</tr>
<tr>
<td>float f;</td>
</tr>
<tr>
<td>int i;</td>
</tr>
<tr>
<td>long l;</td>
</tr>
<tr>
<td>short s;</td>
</tr>
</tbody>
</table>

#### Recall: `char < short < int < long < float < double`

Arithmetic expressions have type at least `int`.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>c + c</code></td>
<td><code>int</code></td>
</tr>
<tr>
<td><code>c + s</code></td>
<td><code>int</code></td>
</tr>
<tr>
<td><code>c - s/i</code></td>
<td><code>int</code></td>
</tr>
<tr>
<td><code>l * 2.0 - i</code></td>
<td><code>double</code></td>
</tr>
<tr>
<td><code>l + 3</code></td>
<td><code>long</code></td>
</tr>
<tr>
<td><code>c + 5.0</code></td>
<td><code>double</code></td>
</tr>
<tr>
<td><code>i/l * 2.0</code></td>
<td><code>double</code></td>
</tr>
<tr>
<td><code>f * 7 - i</code></td>
<td><code>float</code></td>
</tr>
<tr>
<td><code>d + c</code></td>
<td><code>double</code></td>
</tr>
<tr>
<td><code>f * d - l</code></td>
<td><code>double</code></td>
</tr>
</tbody>
</table>

- Both operands promoted to `int`
- `i/l` performs integer division
Assignment Expressions

❖ **Expression**

| x | x+1 | 2*pi*r |

❖ **Assignment expression:** Variable = expression;

\[
\begin{align*}
x &= y; & x &= x+1; & \text{circumference} &= 2 \times \pi \times r;
\end{align*}
\]

❖ **Evaluates** expression on right, **stores result** in variable on left

❖ Like standard expressions (x + y), assignment expressions (x = x+1;) also have a **value** -- the assignment to the left-hand side

❖ **Allows** **chain assignments:**

\[
\begin{align*}
x &= (y = 3); & y &= 3; & x &= 3; & x &= 3
\end{align*}
\]

❖ **For that reason (**!**), assignment expressions evaluated right-to-left**

hence can write: \[x = y = 3;\]

Or: \[a = b = c = 0 \quad \Rightarrow \quad a = b = (c = 0) \quad \Rightarrow \quad a = b = 0 \quad \Rightarrow \quad a = (b = 0) \quad \Rightarrow \quad a = 0\]

\[c = 0 \quad \Rightarrow \quad b = 0 \quad \Rightarrow \quad a = 0\]
Value mismatch

❖ What if expression type (on right) differs from variable type (on left)?
❖ Type conversion: expression (right) converted to the variable type (left)
❖ If larger type converted to smaller (double $\rightarrow$ int)

information may be lost!

```c
int x; int y = 4; double z;

x = y * 3.6;          \[\text{double 14.4}\]
\[\text{int 14}\]

x = y / -5.0;         \[\text{double -0.8}\]
\[\text{int 0}\]

z = y / 5;            \[\text{int 0}\]
\[\text{double 0.0}\]
```

conversion to int always truncates
Outline

❖ Operations

❖ Expressions

❖ More Arithmetic Operations

❖ Logical Operations
Outline

❖ Operations

❖ Expressions

❖ More Arithmetic Operations

❖ Logical Operations
Type Casting

- Often need explicit *type conversion* - e.g. roundoff, $2/3$
- *Casting operator* `(type) expr`, converts `expr` to `type`

```
(int) 2.8 2
(int) -0.9 0
```

- Unary operator
- Has highest precedence in sequence of operations

```
int i = 1, j = 2;
double x, y = 3.14;
x = i / j;          0.0
x = (double)i / j; 0.5
x = i / (double)j;  0.5
x = (double)(i / j);  0.0
x = (int) y;     3.0
```
Roundoff

❖ **(int) rounds toward 0 (truncates)**

❖ How to round to nearest integer?

❖ If just want to print

❖ If want to store?

❖ Solution (for **positive** numbers):

❖ **Negative** numbers, round-up, and 2.5 → 2 -- *later*
Fahrenheit and Celsius

Daniel Gabriel Fahrenheit (1686 - 1736)

\[
\frac{\text{body temp} - \text{brine freeze}}{\text{water freeze} - \text{brine freeze}} \approx 3 = \frac{96}{32} \quad 32 = 2^5
\]

- Set brine fr=0, water fr=32 → body tm=96, water bl=205
- Later: if brine freezes 0 water 32, body ~98.6 (&varies)
- Kept water freeze 32, fixed water boil: 212

Andres Celsius (1701 - 1744)

- Water freeze: 0, water boil: 100

Countries using Celsius:
- Germany, France, Russia, China, Vietnam, Egypt...

Countries using Fahrenheit:
- USA, Jamaica, Belize!
Fahrenheit to Celsius Conversion

\[ T_C = \frac{B_C - F_C}{B_F - F_F} \times (T_F - F_F) + F_C = \frac{100 - 0}{212 - 32} \times (T_F - 32) + 0 = \frac{5}{9} \times (T_F - 32) \]

\[ T_C = \text{slope} = \frac{B_C - F_C}{B_F - F_F} \]

\( T_C, T_F \) — temperature
\( F_F, F_C \) — freezing temp of water
\( B_F, B_C \) — boiling temp of water
```c
#include <stdio.h>

int main() {

    int frizCel = 0, frizFahr = 32;
    int boilCel = 100, boilFahr = 212;
    double tempCel, tempFahr;

    printf("Fahrenheit: ");
    scanf("%lf", &tempFahr);

    tempCel = (boilCel - frizCel)/(boilFahr - frizFahr) * (tempFahr - frizFahr) + frizCel;

    printf("%.1f deg F = %.1f deg C\n", tempFahr, tempCel);

    return 0;
}
```

The formula used to convert Fahrenheit to Celsius is:

\[ T_C = \frac{B_C - F_C}{B_F - F_F} \times (T_F - F_F) + F_C \]
#include <stdio.h>

int main() {
    double frizCel = 0., frizFahr = 32.;
    double boilCel = 100., boilFahr = 212.;
    double tempCel, tempFahr;

    printf("Fahrenheit: ");
    scanf("%lf", &tempFahr);

    tempCel = (boilCel - frizCel)/(boilFahr - frizFahr) * (tempFahr - frizFahr) + frizCel;

    printf("%.1f deg F = %.1f deg C\n", tempFahr, tempCel);
    return 0;
}
```c
#include <stdio.h>

int main() {
    int frizCel = 0, frizFahr = 32;
    int boilCel = 100, boilFahr = 212;
    double tempCel, tempFahr;

    printf("Fahrenheit: ");
    scanf("%lf", &tempFahr);

    tempCel = (double)(boilCel - frizCel)/(boilFahr - frizFahr)
               * (tempFahr - frizFahr) + frizCel;

    printf("%.1f deg F = %.1f deg C\n", tempFahr, tempCel);

    return 0;
}
```

The corrected code snippet calculates the temperature in Celsius `T_C` based on the formula:

\[ T_C = \frac{B_C - F_C}{B_F - F_F} \times (T_F - F_F) + F_C \]
Abbreviations

❖ Often, need expressions like \( x = x + y \)

❖ **Shorthand:** \( x += y \)

❖ Works for all arithmetic operations:

\[
\begin{align*}
    x &= x + y \\
    x &= x - 3 \\
    x &= x \times x \\
    x &= x / 5.3 \\
    x &= x \% 4
\end{align*}
\]

\[
\begin{align*}
    x &= x+y \\
    x &= x-3 \\
    x &= x^2 \\
    x &= x/5.3 \\
    x &= x\%4
\end{align*}
\]

❖ Also works on logic operations (&&, ||) - soon
Unary Increment and Decrement

- Increasing or decreasing a variable by 1 very common
- Even shorter shorthand
  
  - Increasing:
    - `x = x + 1`
    - `x += 1`
    - `x++`
  
  - Decreasing:
    - `x = x - 1`
    - `x -= 1`
    - `x--`

- Why both `x++` and `++x`?
  
  - `x++` increments `x` after expression evaluated
  - `++x` increments `x` before expression evaluated

Example: `f(x++)` vs `f(x); x++`

```
x = 5;
y = ++x;
z = x++;
```

```
x  y  z
5  6  6
6  6  6
```

```
increment.c
```
### Precedence and Associativity Table

<table>
<thead>
<tr>
<th>Operation</th>
<th>Notation</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parentheses</td>
<td>(  )</td>
<td>left to right</td>
</tr>
<tr>
<td>Unary Operations</td>
<td>++  --  +  -</td>
<td>right to left</td>
</tr>
<tr>
<td>Multiplication and Division</td>
<td>*   /   %</td>
<td>left to right</td>
</tr>
<tr>
<td>Addition and Subtraction</td>
<td>+   -</td>
<td>left to right</td>
</tr>
<tr>
<td>Assignment Operations</td>
<td>=  +=  -= *= /= %=</td>
<td>right to left</td>
</tr>
</tbody>
</table>

**Example:** Suppose that the value of $z$ is 6. Then...

```
x = y = 4 + z++ * 5 / (2 + 1)
```

```
Z ← 7

x, y ← 15
```

Increasing precedence

```
z * z++
++z * z++
```

possible warning, best avoid
Outline

❖ Operations
❖ Expressions
❖ More Arithmetic Operations
❖ Logical Operations
Outline

❖ Operations

❖ Expressions

❖ More Arithmetic Operations

❖ Logical Operations
Logical Operators

❖ Three logical operators: **NOT, AND, OR**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>&amp;&amp;</td>
</tr>
<tr>
<td>OR</td>
<td></td>
</tr>
<tr>
<td>NOT</td>
<td>!</td>
</tr>
</tbody>
</table>

❖ Defined via truth tables to reflect normal meaning of NOT, AND, OR

| exp1 | exp2 | AND exp1 && exp2 | OR exp1 || exp2 | NOT !exp1 |
|------|------|------------------|-----------|----------|-----------|
| F    | F    | F                | F         | T        |
| F    | T    | F                | T         | T        |
| T    | F    | F                | T         | F        |
| T    | T    | T                | T         | F        |

❖ Operators can be combined to construct complex expressions
Logical Expressions

| exp1 | exp2 | exp1 && exp2 | exp1 || exp2 | !exp1 |
|------|------|-------------|-------------|-------|
| F    | F    | F           | F           | T     |
| F    | T    | F           | T           | T     |
| T    | F    | F           | T           | F     |
| T    | T    | T           | T           | F     |

- Operators combined to construct complex expressions:
- Evaluated left to right
- && before ||
- Parentheses modifies
- Unary ! before &&
- Evaluation stops when value determined

Any logic statement (propositional formula) can be written in terms of **AND**, **OR**, and **NOT**.
Boolean Values

❖ Type `_Bool` specifies a Boolean (0/1) variable
  ◦ 0 - false
  ◦ 1 - true

  ```c
#include <stdio.h>

int main() {
    _Bool a=0, b=1;
    printf("a=%d b=%d a&&b=%d a||b=%d\n", a, b, a&&b, a||b);
    return 0;
}
```

❖ `<stdbool.h>` library (header file) defines
  ◦ `bool` - `_Bool`
  ◦ `false` - 0
  ◦ `true` - 1

  ```c
#include <stdio.h>
#include <stdbool.h>

int main() {
    bool a=false, b=true;
    printf("a=%d b=%d a&&b=%d a||b=%d\n", a, b, a&&b, a||b);
    return 0;
}
```

❖ Still Occupies 1 byte of memory

```
#include <stdio.h>

int main() {
    int main() {
    _Bool a=0, b=1;
    printf("a=%d b=%d a&&b=%d a||b=%d\n", a, b, a&&b, a||b);
    return 0;
}
```

```
#include <stdio.h>
#include <stdbool.h>

int main() {
    bool a=false, b=true;
    printf("a=%d b=%d a&&b=%d a||b=%d\n", a, b, a&&b, a||b);
    return 0;
}
```
Logical Values of Numbers

❖ Every **number** has a *logical value*:

- Zero: False
- Nonzero: True

❖ Useful in conditionals (next lecture):

```c
if(expression){...} else {...}
```

**Example**

```c
int a = 3;
int b = -3;
double c = 1.0;
```

<table>
<thead>
<tr>
<th>Expression</th>
<th>Numerical value</th>
<th>Logical Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>3</td>
<td>True</td>
</tr>
<tr>
<td>a + b</td>
<td>0</td>
<td>False</td>
</tr>
<tr>
<td>a += b</td>
<td>0</td>
<td>False</td>
</tr>
<tr>
<td>a = 2*c</td>
<td>2</td>
<td>True</td>
</tr>
<tr>
<td>b + 3*c</td>
<td>0.0</td>
<td>False</td>
</tr>
<tr>
<td>b &amp;&amp; c</td>
<td>1</td>
<td>True</td>
</tr>
</tbody>
</table>

Zero values of **all types** (0, 0.0, ‘\0’) are False

**True represented as 1**
Relational Operators

❖ 2 operators for testing equality (or inequality) **==** **!=**

❖ 4 operators for testing relations (greater, smaller) **<** **<=** **>** **>=**

❖ value 1 (true) if condition holds

0 (false) if condition does not hold

<table>
<thead>
<tr>
<th>Expression</th>
<th>Numerical Value</th>
<th>Logical Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>d = c</td>
<td>3</td>
<td>True</td>
</tr>
<tr>
<td>c = a</td>
<td>0</td>
<td>False</td>
</tr>
<tr>
<td>a == b</td>
<td>0</td>
<td>False</td>
</tr>
<tr>
<td>a != b</td>
<td>1</td>
<td>True</td>
</tr>
<tr>
<td>b &lt;= 8</td>
<td>1</td>
<td>True</td>
</tr>
<tr>
<td>0 &lt; 4 &lt; 3</td>
<td>1</td>
<td>True</td>
</tr>
<tr>
<td>0 &lt; .5 &lt; 1</td>
<td>0</td>
<td>False</td>
</tr>
</tbody>
</table>

To check \( a < b < c \) : \( a < b && b < c \)

**Note**

= assignment
== test equality

Evaluated left-to-right
Unlike arithmetic, value is often decided from part of expression

Evaluation **stops** when logical value can be determined

Example

```c
if (age <= 90) && (GPA > 3.5 || income >= 1000000)
    plan_wedding = true;
```

```c
if (a/b > 1) a++;
/* What happens if b is zero? */
```

```c
if ((b != 0) && (a/b > 1)) a++;
/* Will never crash! */
```
Precedence and Associativity Table

- ! is unary with same high precedence as other unary operators
- Relational (<) and equality (==) operators after +,-
- && and || follow

<table>
<thead>
<tr>
<th>Operation</th>
<th>Notation</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parentheses</td>
<td>( )</td>
<td>left to right</td>
</tr>
<tr>
<td>Unary Operations</td>
<td>++ -- - +</td>
<td>right to left</td>
</tr>
<tr>
<td>! (type) sizeof</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multiplication and Division</td>
<td>* / %</td>
<td>left to right</td>
</tr>
<tr>
<td>Addition and Subtraction</td>
<td>+ -</td>
<td>left to right</td>
</tr>
<tr>
<td>Relational Operators</td>
<td>&lt; &lt;= &gt; &gt;=</td>
<td>left to right</td>
</tr>
<tr>
<td>Equality Operators</td>
<td>== !=</td>
<td>left to right</td>
</tr>
<tr>
<td>Logical AND</td>
<td>&amp;&amp;</td>
<td>left to right</td>
</tr>
<tr>
<td>Logical OR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Assignment Operations</td>
<td>= += -= *= /= %=</td>
<td>right to left</td>
</tr>
</tbody>
</table>