EECS 1710
Programming for Digital Media

Week 2 :: Programming Basics

YORK UNIVERSITÉ UNIVERSITY

This Week

Lecture 3 (continued):

- Real number types (float/double)
- Integer-based vs float-based
- Assignment
- boolean & char types
- Basic numeric operators: +, -, *, /
- Division caveats ☺

Numeric types

`int`, `long`, **`float`**, **`double`**, etc.

Real (decimal) types

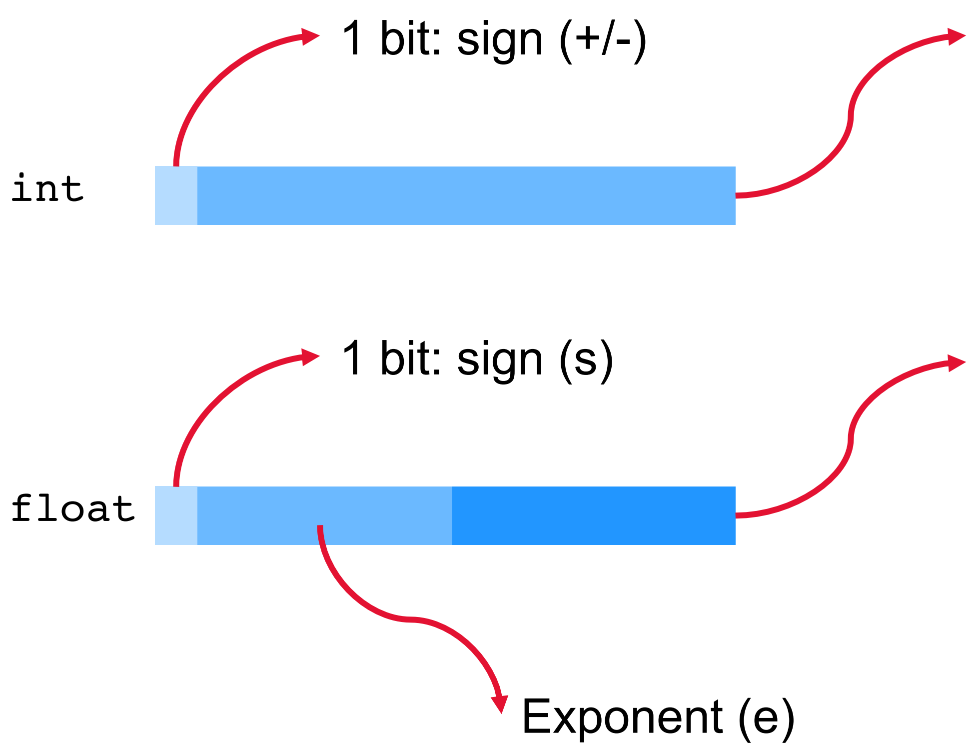
Reals (format, storage, range)

- Format
  - Formatted according to the IEEE-754 standard for floating point arithmetic
  - Includes a fractional part and a power

- Storage
  - `float` → 4 bytes
  - `double` → 8 bytes
- Range
  - `float` → $\pm 10^{38}$ with 7 significant digits
  - `double` → $\pm 10^{308}$ with 15 significant digits

How can `float` and `int` encode different ranges using same number of bits??

- Answer:
  - Different representations! (i.e. bits configured differently)



$(-1)^s * (m) * B^e$

where B = base (e.g. 2 or 10)

Assignment

- The statement (from `Area.pde`)

```
rectWidth = 8;
```

is of the general form

```
name = value;
```

- Pre-declared and in-scope
- Type can hold RHS
- Content will be overwritten

- Literal
- Name, or
- Expression

Note: RHS = right-hand side, LHS = left-hand side

Assigning Literals to Real Types

```
double x;

double interestRate = 1.5;

float z = -1.1f;

double abc = 3.4E-5;
```

Float literal (default is double)

Same as

0.000034

$= 3.4 \times 10^{-5}$

Assignment

Examples

```
int quantity;
quantity = 25;
```

Declaration

Assignment

```
int quantity = 25;
int stock = quantity;
```

Declaration and assignment combined

Name of variable on RHS

```
int quantity = 25;
char grade = 'B';
boolean isFound = false;
double intRate = 1.25;
```

Expression on RHS

```
int stock = 100;
int order = 15;
int total = order + stock;
```

Demo 1 (using integers & reals)

```
// Block1: integer types

{
  int quantity;
  quantity = 25;

  int stock = quantity;
  // long stock = quantity;

  println("block1:");
  print("quantity = ");
  println(quantity);
  print("stock = ");
  println(stock);
  println();
}
```

```
// Block2: real numeric types

{
  float quantity;
  quantity = 25.0;
  float stock = quantity;

  println("block2:");
  print("quantity = ");
  println(quantity);
  print("stock = ");
  println(stock);
  println();
}
```

Question:

- a float value cannot be assigned to an int, but an int can be assigned to a float! Why?

A float can represent a whole number (with zero values in its decimal places), but a decimal number cannot be represented by an int (it has no way to encode the decimal places)

Special Cases

- What happens if…
  - Division by zero
    - Integers: throws an arithmetic exception
    - Reals: assigns a fictitious value, `NaN` (“not a number”)
  - Out of range result
    - Integers: range is treated as circular
    - Reals: assigns a fictitious value, `Infinity`

The Boolean Type (boolean)

- Stores the result of a condition
- Has only two possible values, `true` or `false`

  (can think of this as a pure binary type)
- `true` and `false` are reserved words
- Boolean variables are not integers !

- Declaration & Assignment:

```
boolean myBool;
myBool = true;
myBool = false;
```

The Character Type (char)

- A `char` is a letter, digit, or symbol
- Examples:

  1 @ a A b B & *] { ~ %

- Stores a code for a character, not the typeface itself
- The codes for English use ASCII1
- `char` is stored as an (unsigned) integer type

- Numeric coding of characters uses the *Unicode* character set
- Unicode has 64K codes (see following slides)

[1] ASCII codes are the first 256 entries in the Unicode character set.
Try Wikipedia for more details.

Unicodes

Decimal	Unicode (U + hex)	Content
0–31	\u0000 - \u001f	control characters
32	\u0020	space
48–57	\u0030 - \u0039	the digits 0 to 9
65–90	\u0041 - \u005a	uppercase letters A–Z
97–122	\u0061 - \u007a	lowercase letters a–z

Decimal	Unicode	Escape Sequence	Character
9	\u0009	\t	HT: horizontal tab
10	\u000a	\n	LF: line feed
12	\u000c	\f	FF: form feed
13	\u000d	\r	CR: carriage return
32	\u0020		SP: space

32	\u0020	SP
33	\u0021	!
34	\u0022	"
35	\u0023	#
36	\u0024	$
37	\u0025	%
38	\u0026	&
39	\u0027	'
40	\u0028	(
41	\u0029	)
42	\u002a	*
43	\u002b	+
44	\u002c	,
45	\u002d	-
46	\u002e	.
47	\u002f	/
48	\u0030	0
49	\u0031	1
50	\u0032	2
51	\u0033	3
52	\u0034	4
53	\u0035	5
54	\u0036	6
55	\u0037	7
56	\u0038	8
57	\u0039	9
58	\u003a	:
59	\u003b	;
60	\u003c	<
61	\u003d	=
62	\u003e	>
63	\u003f	?

64	\u0040	@
65	\u0041	A
66	\u0042	B
67	\u0043	C
68	\u0044	D
69	\u0045	E
70	\u0046	F
71	\u0047	G
72	\u0048	H
73	\u0049	I
74	\u004a	J
75	\u004b	K
76	\u004c	L
77	\u004d	M
78	\u004e	N
79	\u004f	O
80	\u0050	P
81	\u0051	Q
82	\u0052	R
83	\u0053	S
84	\u0054	T
85	\u0055	U
86	\u0056	V
87	\u0057	W
88	\u0058	X
89	\u0059	Y
90	\u005a	Z
91	\u005b	[
92	\u005c	\
93	\u005d	]
94	\u005e	^
95	\u005f	_

96	\u0060	`
97	\u0061	a
98	\u0062	b
99	\u0063	c
100	\u0064	d
101	\u0065	e
102	\u0066	f
103	\u0067	g
104	\u0068	h
105	\u0069	i
106	\u006a	j
107	\u006b	k
108	\u006c	l
109	\u006d	m
110	\u006e	n
111	\u006f	o
112	\u0070	p
113	\u0071	q
114	\u0072	r
115	\u0073	s
116	\u0074	t
117	\u0075	u
118	\u0076	v
119	\u0077	w
120	\u0078	x
121	\u0079	y
122	\u007a	z
123	\u007b	{
124	\u007c	|
125	\u007d	}
126	\u007e	~
127	\u007f	

More complete set: https://www.rapidtables.com/code/text/unicode-characters.html

Declaration & assignment of characters:

- Character literals are recognized by single quotes surrounding a character, e.g., `'A'`
- Special characters, such a single quote itself, are represented as literals using *escape sequences*

Escape	Meaning
`\uxxxx`	The character whose code is (hex) xxxx
`\'`	Single quote
`\"`	Double quote
`\\`	Backslash
`\n`	New line
`\r`	Carriage return
`\f`	Form Feed
`\t`	Tab
`\b`	Backspace

```
// declaration
   char myChar;

// standard characters
   myChar = ‘a’;
   myChar = ‘A’;
   myChar = ‘$’;
   myChar = ‘)’;
   myChar = ‘>’;

// using escape characters
   myChar = ‘\’’;          // single quote ‘
   myChar = ‘\”’;          // double quote “
   myChar = ‘\\’;          // backslash /
   myChar = ‘\n’;          // new line

// using unicodes
   myChar = ‘\u0061’;      // ’a’
   myChar = ‘\u0041’;      // ‘A’
   myChar = ‘\u0024’;      // ‘$’
   myChar = ‘\u007c’;      // ‘[‘
   myChar = ‘\u0151’;      // ‘ő’
   myChar = ‘\u03A3’;      // ‘Σ’
```

```
// Block3: booleans and chars

{
  char grade = 'B';
  char exclaim = '\u0021';
  boolean isFound = false;

  println("block3:");
  print("grade = ");
  print(grade); println(exclaim);

  int gradeNum = grade;
  print("gradeNum = ");
  println(gradeNum);

  println();
  print("isFound = ");
  println(isFound);


}
```

Primitive types (summary)

PRIMITIVE TYPES			Type	Size (bytes)	Approximate Range min	max	S.D.
NUMBER	INTEGER	SIGNED	byte	1	-128	+127	-
			short	2	-32,768	+32,767	-
			int	4	-2×10^{9}	$+2\times10^{9}$	-
			long	8	-9×10^{18}	$+9\times10^{18}$	-
		UNSIGNED	char	2	0	65,535	-
	REAL	SINGLE	float	4	$+3.4\times10^{38}$	$+3.4\times10^{38}$	7
		DOUBLE	double	8	-1.7×10^{308}	$+1.7\times10^{308}$	15
BOOLEAN			boolean	1	true/false		-

```
// Block4: exploring range limits

{
  int stock = 65536;
  int order = 1002314;
  //int stock = -2147483648;
  //int order = -1;

  int total = order + stock;

  println();
  println("block4:");

  print("total = "); print(stock); print(" + "); print(order);
  print(" = "); println(total);

  float f1 = 3.4e38;
  float f2 = 1000;
  float f3 = f1 * f2;

  println();
  print("f1 = "); println(f1);
  print("f2 = "); println(f2);
  print("f3 = f1+f2 = "); println(f3);
}
```

Expressions & Operators

- *Expressions* involve one or more data values that appear together with *operators*
- *Operators* define specific actions on data
- *Operators* are usually specific to a given type
  - E.g. standard operators + - * / in general, work on integer and real types
  - Their function may differ slightly depending on the type they are operating on

- Expressions are typically processed from left to right (though there are exceptions that give some operators precedence over others)
- Parenthesis in an expression can override operator precedence

`int` arithmetic operators (summary)

Precedence	Operator	Kind	Syntax	Operation
-5 →	+	infix	x + y	add y to x
	-	infix	x - y	subtract y from x
-4 →	*	infix	x * y	multiply x by y
	/	infix	x / y	divide x by y
	%	infix	x % y	remainder of x / y
-2 ←	+	prefix	+x	identity
	-	prefix	-x	negate x
	++	prefix	++x	x = x + 1; result = x
	--	prefix	--x	x = x - 1; result = x
-1 →	++	postfix	x++	result = x; x = x + 1
	--	postfix	x--	result = x; x = x - 1

Lowest priority

Highest priority

Notes (1)

- Division (/)
  - For integer operands, the result is an integer rounded toward zero, so

```
 5 / 4 → 1
-5 / 4 → -1
```

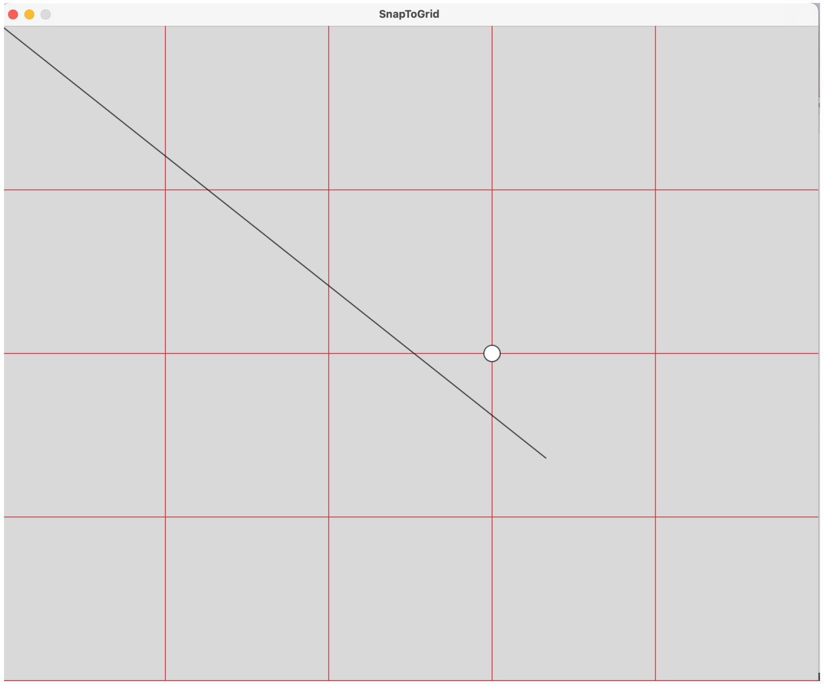
  - For real operands, the result is a real

```
 5.0 / 4.0 →  1.25
-5.0 / 4.0 → -1.25
```

Example

```
// block 5: division caveats

{
  int oneHour = 60;       // mins
  int onePres = 20;       // 20 mins per pres

  int presPerHour = oneHour/onePres;

  print("oneHour = "); print(oneHour); println(" mins");
  print("onePres = "); print(onePres); println(" mins");
  print("presPerHour = "); print(presPerHour); println();

}
```

Drawing Example:




```
// snap to grid – exploits integer division

// grid vertical and horizontal spacing
int hSpacing = 200;
int vSpacing = 200;

void setup() {
  size(1000,1000);
}

void draw() {
  background(0xd9d9d9);    // light gray

  // grid pattern
  stroke(#c63e3e);
  line(200,0,200,height);  line(0,200,width,200);
  line(400,0,400,height);  line(0,400,width,400);
  line(600,0,600,height);  line(0,600,width,600);
  line(800,0,800,height);  line(0,800,width,800);
  stroke(0,0,0);

  // calculate a snapTo point based on mouse position
  float snapPointX = (mouseX/hSpacing)*hSpacing;
  float snapPointY = (mouseY/vSpacing)*vSpacing;

  line(0, 0, mouseX, mouseY);
  circle(snapPointX,snapPointY,20);
}
```