Working With Data: Variables and Expressions

ISE 108: Introduction to Programming

Stony Brook University

1

Variables

- Variables are the “nouns” of a program
  - They represent pieces of information
- Variables come in different *types*:
  - `int` and `long` store integers
  - `char` stores single characters (‘a’)
  - `float` and `double` store decimal values

2

Declarations

- Processing requires you to declare a variable before you use it
  - This tells Processing what type of behavior to expect from that variable
- Declare a variable (tell the compiler that you’re going to use it) by writing the type, followed by the name:

```
int x; // Declares an integer variable x
double pi; // pi holds a fractional number
```

3

Variable Scope

- Declaring variables comes with a catch: scope
- A variable’s *scope* is the region of a program in which it is defined (visible)
- Scope is restricted to the smallest set of (matched) curly braces that surrounds a variable
  - This may be just one function, or the entire program

4

Storing Values in Variables

- Use a *single* '=' to store a value in a variable
  - "=" translates to "is assigned the value"
  - The left gets the value on the right side:

```
x = 5; // Stores the integer value 5 in x
```

- We can combine assignments with declarations:

```
char foo = 'g';
```

- 5 and 'g' are *literals* (actual values)

5

Constants

- A *constant* is a value that cannot change
- To declare a constant, use the keyword `final`:

```
final double PI = 3.1415926;
```

- By convention, constant names are capitalized
- Strings (sequences of characters enclosed in double quotes) are automatically constants.

6

Arithmetic

- Basic arithmetic (addition, subtraction, multiplication, division) uses the normal order of operations
- Dividing two integers only gives you the quotient
  - Use modulus (%) to get the remainder
    - e.g., 12 / 7 is 1, and 12 % 7 is 5
  - The modulus operation works like "clock arithmetic"

7

The Rules of Arithmetic

- In Processing, arithmetic follows two basic rules:
  1. Both operands must be the same type
  2. The answer has the same type as the operands
- If the operands are of different types (e.g., int and float), then they must be converted to the same type first
  - Smaller types are automatically promoted to larger types

char << int << long << float << double

8

Type Conversion

- A value can always be stored in a variable of a larger (further right) type
  - Ex. an `int` can be stored in a `float`
- If you want to go the other way (to store a value in a smaller type), you ***must*** explicitly cast (convert) the value to the desired type:

```
int x = (int) 3.14159;
```

- To cast, precede the value with the new type in parentheses
- Casting may (and often does) lose data; the original value will be truncated (for example, a decimal will lose the part after the point).
- **Note:** Casting a value to an integer **does not** round the resulting value

9

Conversion Examples

```
double average = 100.0 / 8.0;

average = 100.0 / 8;

average = 100 / 8;
```

100.0 and 8.0 are both `doubles`; the result (12.5) is the same type as `average`.

100.0 and 8 are different types; 8 is promoted to 8.0, and we proceed as before.

100 and 8 are both integers, so we perform integer division to get 12. This is upcast to 12.0, because average is of type `double`.

10

Conversion Examples

```
int sumGrades = 100.0 / 8.0

sumGrades = (int) 100.0 / 8.0

sumGrades = (int) (100.0 / 8.0)
```

Both operands are `doubles`; their quotient (12.5) is also a `double`. The `double`-to-`int` cast must be explicit, so the compiler will report an error.

100.0 (a `double`) is cast to an int, but 8.0 is still a `double` (casting is done before arithmetic). 100 is promoted back to 100.0, and the problem remains unchanged.

In the last case, 100.0 and 8.0 are divided first. The result (12.5) is cast from `double` to an `int`, losing the .5, and is stored in `sumGrades` as 12 (no rounding!).

11

Conversion Examples

```
double fiftyPercent = 50 / 100;

fiftyPercent = 50.0 / 100.0;
```

50 and 100 are both integers; dividing them produces 0. 0 is converted to a `double` to match `fiftyPercent`'s type, and the result is 0.0.

50.0 and 100.0 are both `doubles`; dividing them gives 0.5 (a `double`). This is stored unchanged in `fiftyPercent`.

12

Built-in Variables

- Processing includes a few special variables that can be used for user interaction
  - mouseX — contains the current mouse x coordinate
  - mouseY — contains the current mouse y coordinate
    - pmouseX and pmouseY hold the previous coordinates
  - key — stores the value of the last key pressed by the user

Next Time

- Conditional statements
  - `if...else`
  - `switch` statements
- Repetition statements
  - `for`, `while`, and `do...while` loops