University of British Columbia
CPSC 111, Intro to Computation
Jan-Apr 2006

Tamara Munzner

Data Types, Assignment, Expressions, Constants

Lecture 3, Thu Jan 12 2006

based on slides by Kurt Eiselt

http://www.cs.ubc.ca/~tmm/courses/cpsc111-06-spr

News

- Weekly Question 1 due today

- Labs and tutorials started this week
  - Labs on Friday cancelled
    - you've been reassigned elsewhere
    - if you missed assigned lab this week, attend another session if possible

Reminder: Reading This Week

- Ch 1.1 - 1.2: Computer Anatomy
  - from last time

- Ch 1.3 – 1.8: Programming Languages
- Ch 2.1-2.2, 2.5: Types/Variables, Assignment, Numbers
- Ch 4.1-4.2: Numbers, Constants

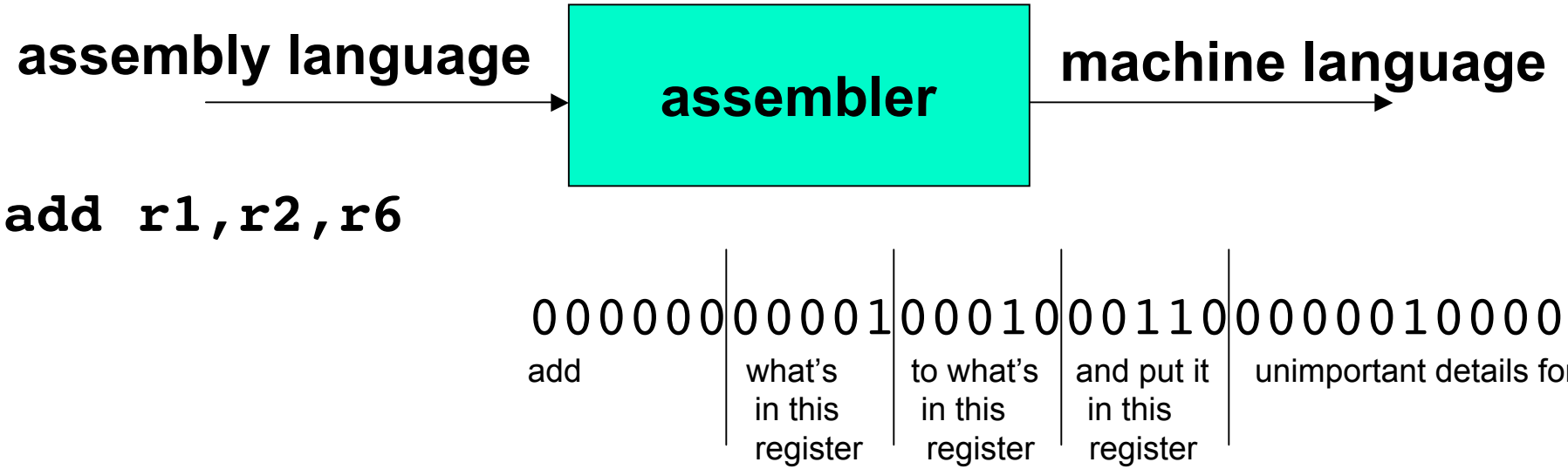
Reading for Next Week

- Rest of Chap 2
  - 2.3-4, 2.6-2.10
- Rest of Chap 4
  - 4.3-4.7

Objectives

- Understand how to declare and assign variables
- Understand when and how to use which data type
- Understand how to convert between data types
- Understand how to interpret expressions
- Understand when to use constants

Recap: Assembly and Machine Languages



- Hard to read, write, remember
- Many instructions required to do things
- Different languages for each computer type

Recap: High-Level Languages

- Program written in high-level language converted to machine language instructions by another program called a compiler (well, not always)

- High-level instruction: A = B + C
  becomes at least four machine language instructions!

```
00010000001000000000000000000010   load B
00010000001000000000000000000011   load C
00000000001000100011000000100000   add them
00010100110000000000000000000001   store in A
```

Recap: Sample Java Program

- Comments, whitespace ignored by compiler

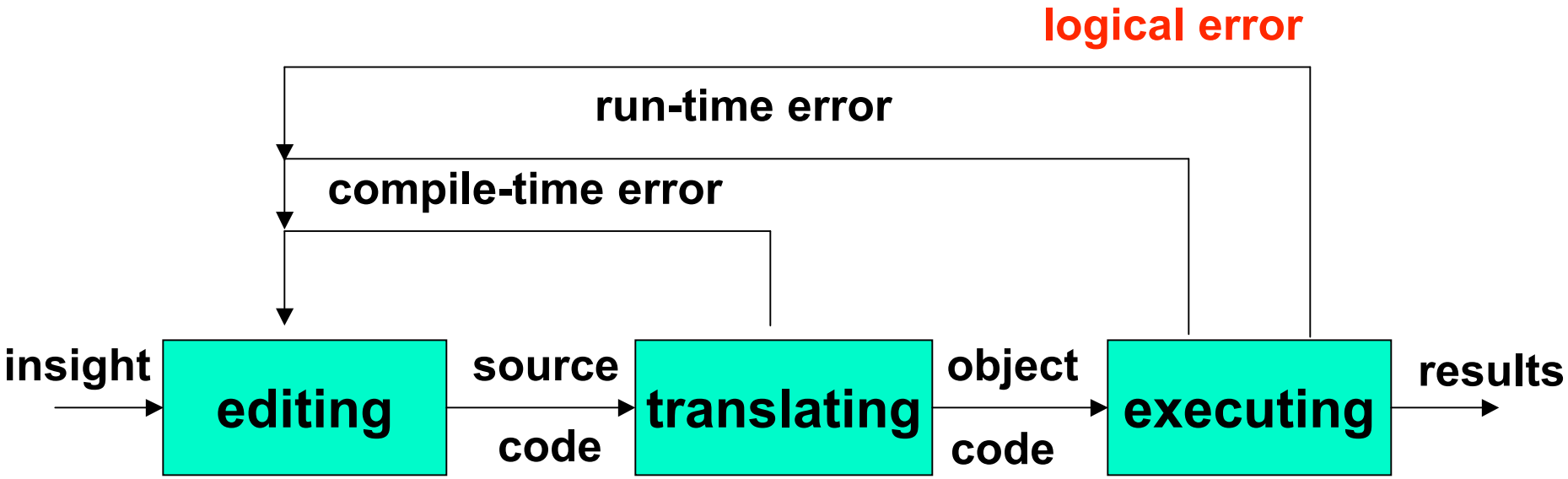
```
//*************************************************
// Oreo.java          Author:  Kurt Eiselt
//
// Demonstrating simple Java programming concepts while
// revealing one of Kurt's many weaknesses
//*************************************************

public class Oreo
{
  //*************************************************
  // demand Oreos
  //*************************************************
  public static void main (String[] args)
  {
    System.out.println ("Feed me more Oreos!");
  }
}
```

Recap: Identifiers

- Identifiers: start with letter [a-Z,$,_], then letters of digits [0-9]
  - and not be reserved words
  - case matters
  - meaningful and descriptive, yet concise

Recap: Errors



- Compile-time errors
  - syntax/structure
- Run-time errors
- Logical errors
  - semantics/meaning

Recap: Variables

- Variable: name for location in memory where data is stored
  - avoid having to remember numeric addresses
  - like variables in algebra class

- Variable names begin with lower case letters
  - Java convention, not compiler/syntax requirement

Recap: Data Types

- Java requires that we tell it what kind of data it is working with
- For every variable, we have to declare a data type
- Java language provides eight primitive data types
  - i.e. simple, fundamental
- For more complicated things, can use data types
  - created by others provided to us through the Java libraries
  - that we invent
    - More soon - for now, let's stay with the primitives
- We want `a`, `b`, and `c` to be integers
  - Here's how we do it...

Recap: Variables and Data Types

```
//*****************************************
// Test3.java        Author: Kurt
//
// Our third use of variables!
//*****************************************

public class Test3
{
    public static void main (String[] args)
    {
        int a;  //these
        int b;  //are
        int c;  //variable declarations
        b = 3;
        c = 5;
        a = b + c;
        System.out.println ("The answer is " + a);
    }
}
```

Variable Declaration and Assignment

- variable declaration is instruction to compiler
  - reserve block of main memory large enough to store data type specified in declaration
- variable name is specified by identifier
- syntax:
  - *typeName variableName;*

Data Types: Int and Double

- int
  - integer
- double
  - real number
  - (double-precision floating point)

Floating Point Numbers

- significant digits
  - 42
  - 4.2
  - 42000000
  - .000042

Floating Point Numbers

- significant digits
  - 42 $= 4.2 * 10 = 4.2 * 10^{1}$
  - 4.2 $= 4.2 * 1 = 4.2 * 10^{0}$
  - 42000000 $= 4.2 * 10000000 = 4.2 * 10^{7}$
  - .000042 $= 4.2 * .00001 = 4.2 * 10^{-5}$

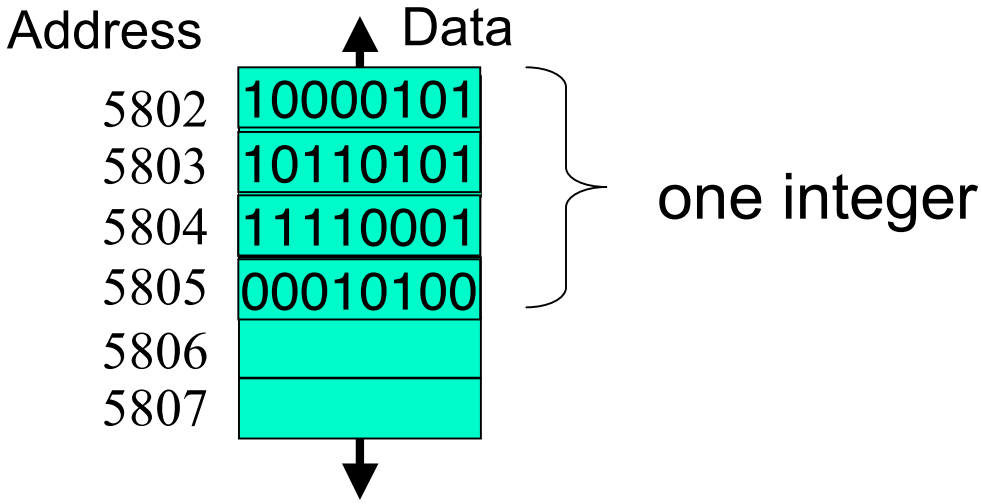
Floating Point Numbers

- significant digits
  - 42 $= 4.2 * 10 = 4.2 * 10^{1}$
  - 4.2 $= 4.2 * 1 = 4.2 * 10^{0}$
  - 42000000 $= 4.2 * 10000000 = 4.2 * 10^{7}$
  - .000042 $= 4.2 * .00001 = 4.2 * 10^{-5}$
- only need to remember
  - nonzero digits
  - where to put the decimal point
    - floats around when multiply/divide by 10

Data Type Sizes

Type	Size	Min	Max
`int`	4 bytes	-2,147,483,648	2,147,483,647
`double`	8 bytes	**approx -1.7E308 (15 sig. digits)**	**approx 1.7E308 (15 sig. digits)**

- fixed size, so finite capacity



Variable Declaration Examples

- person’s age in years
- height of mountain to nearest meter
- length of bacterium in centimeters
- number of pets at home

Assignment

```
//*****************************************
// Test3.java        Author: Kurt
//
// Our third use of variables!
//*****************************************

public class Test3
{
    public static void main (String[] args)
    {
        int a;
        int b;
        int c;
        b = 3;       // these
        c = 5;       // are
        a = b + c;   // assignment statements
        System.out.println ("The answer is " + a);
    }
}
```

Assignment Statements

- Assignment statement assigns value to variable
  - sometimes say binds value to variable
- Assignment statement is
  - identifier
  - followed by assignment operator (=)
  - followed by expression
  - followed by semicolon (;)

```
b = 3;
c = 8;
a = b + c;
weekly_pay = pay_rate * hours_worked;
```

- Note that = is no longer a test for equality!

Assignment Statements

- Java first computes value on right side
- Then assigns value to variable given on left side

```
x = 4 + 7;      // what's in x?
```

- Old value will be overwritten if variable was assigned before

```
x = 2 + 1;      // what's in x now?
```

Assignment Statements

- Here's an occasional point of confusion:

```
a = 7;              // what's in a?
b = a;              // what's in b?
                    // what's in a now???
```

Assignment Statements

- Here's an occasional point of confusion:

```
a = 7;              // what's in a?
b = a;              // what's in b?
                    // what's in a now???
System.out.println("a is " + a + "b is " +b);
```

- Find out! Experiments are easy to do in CS

Assignment Statements

- Here's an occasional point of confusion:

```
a = 7;              // what's in a?
b = a;              // what's in b?
                    // what's in a now???
System.out.println("a is " + a + "b is " +b);
```

- Variable values on left of = are clobbered
- Variable values on right of = are unchanged
  - copy of value assigned to a also assigned to b
  - but that doesn't change value assigned to a

Assignment Statements

- Here's an occasional point of confusion:

```
a = 7;              // what's in a?
b = a;              // what's in b?
                    // what's in a now???
System.out.println("a is " + a + "b is " +b);
a = 8;
System.out.println("a is " + a + "b is " +b);
```

- Memory locations a and b are distinct
  - copy of value assigned to a also assigned to b
  - changing a later does not affect previous copy
    - more later

Variable Declaration and Assignment

- variable declaration is instruction to compiler
  - reserve block of main memory large enough to store data type specified in declaration
- variable name is specified by identifier
- syntax:
  - *typeName variableName;*
  - *typeName variableName = value;*
    - can declare and assign in one step

Expressions

- expression is combination of
  - one or more operators and operands
  - operator examples: +, *, /, ...
  - operand examples: numbers, variables, ...
  - usually performs a calculation
    - don’t have to be arithmetic but often are
  - examples

3
7 + 2
7 + 2 * 5
(7 + 2) * 5

Operator Precedence

- What does this expression evaluate to?

7 + 2 * 5

Operator Precedence

- What does this expression evaluate to?

  7 + 2 * 5

- Multiplication has higher operator precedence than addition (just like in algebra)

precedence	operator	operation
1 higher	+ -	unary plus and minus
2	* / %	multiply, divide, remainder
3 lower	+ -	add, subtract

Operator Precedence

- What does this expression evaluate to?

  7 + 2 * 5

- Multiplication has higher operator precedence than addition (just like in algebra)

precedence	**operator**	**operation**
1 higher	**+ -**	**unary plus and minus**
2	*** / %**	**multiply, divide, remainder**
3 lower	**+ -**	**add, subtract**

- Use parentheses to change precedence order or just clarify intent

  (7 + 2) * 5 **7 + (2 * 5)**

Converting Between Types

- Which of these are legal?
  - `int shoes = 2;`
  - `double socks = 1.75;`
  - `double socks = 1;`
  - `int shoes = 1.5;`

Converting Between Types

- Which of these are legal?
  - `int shoes = 2;`
  - `double socks = 1.75;`
  - `double socks = 1;`
  - `int shoes = 1.5;`
- Integers are subset of reals
  - but reals are not subset of integers

Casting

- Casting: convert from one type to another with information loss
- Converting from real to integer
  - `int shoes = (int) 1.5;`
- Truncation: fractional part thrown away
  - `int shoes = (int) 1.75;`
  - `int shoes = (int) 1.25;`
- Rounding: must be done explicitly
  - `shoes = Math.round(1.99);`

Converting Between Types

```
//*****************************************
// Feet.java   Author: Tamara
// What type of things can be put on feet?
//*****************************************
public class Feet
{
 public static void main (String[] args)
 {
   int shoes = 2;
   int socks = (int) 1.75;
   System.out.println("shoes = " + shoes + " socks = " +
socks);
   int toes = Math.round(1.99);
   System.out.println("toes = " + toes);
 }
}
```

- What's wrong?

Data Type Sizes

Type	Size	Min	Max
`int`	4 bytes	-2,147,483,648	2,147,483,647
`double`	8 bytes	**approx -1.7E308 (15 sig. digits)**	**approx 1.7E308 (15 sig. digits)**

- doubles can store twice as much as ints

Primitive Data Types: Numbers

Type	Size	Min	Max
`byte`	1 byte	-128	127
`short`	2 bytes	-32,768	32,767
`int`	4 bytes	-2,147,483,648	2,147,483,647
`long`	8 bytes	**-9,223,372,036,854,775,808**	**9,223,372,036,854,775,807**
`float`	4 bytes	**approx -3.4E38 (7 sig.digits)**	**approx 3.4E38 (7 sig.digits)**
`double`	8 bytes	**approx -1.7E308 (15 sig. digits)**	**approx 1.7E308 (15 sig. digits)**

- Primary primitives are `int` and `double`
  - three other integer types
  - one other real type

Converting Between Types

```
//*****************************************
// Feet2.java   Author: Tamara
// What type of things can be put on feet?
//*****************************************
public class Feet2
{
 public static void main (String[] args)
 {
   int shoes = 2;
   int socks = (int) 1.75;
   System.out.println("shoes = " + shoes + " socks = " +
socks);
   long toes = Math.round(1.99);
   System.out.println("toes = " + toes);
 }
}
```

Primitive Data Types: Non-numeric

- Character type
  - named char
  - Java uses the Unicode character set so each char occupies 2 bytes of memory.
- Boolean type
  - named boolean
  - variables of type boolean have only two valid values
    - true and false
  - often represents whether particular condition is true
  - more generally represents any data that has two states
    - yes/no, on/off

What Changes, What Doesn't?

```
//*************************************
// Vroom.java Author: Tamara
// Playing with constants
//*************************************
public class Vroom
{
 public static void main (String[] args)
 {
   double lightYears, milesAway;
   lightYears = 4.35; // to Alpha Centauri
   milesAway = lightYears * 186000 *60*60*24*365;
   System.out.println("lightYears: " + lightYears + "
milesAway " + milesAway);
   lightYears = 68; // to Aldebaran
   milesAway = lightYears * 186000 *60*60*24*365;
   System.out.println("lightYears: " + lightYears + "
milesAway " + milesAway);
 }
}
```

Constants

- Things that do not vary
  - unlike variables
  - will never change
- Syntax:
  - final *typeName variableName;*
  - final *typeName variableName = value;*
- Constant names in all upper case
  - Java convention, not compiler/syntax requirement

Programming With Constants

```
public static void main (String[] args)
 {
   double lightYears, milesAway;

   final int LIGHTSPEED = 186000;
   final int SECONDS_PER_YEAR = 60*60*24*365;

   lightYears = 4.35; // to Alpha Centauri
   milesAway = lightYears * LIGHTSPEED * SECONDS_PER_YEAR;
   System.out.println("lightYears:  " + lightYears + "
miles " + milesAway);

   lightYears = 68; // to Aldebaran
   milesAway = lightYears * LIGHTSPEED * SECONDS_PER_YEAR;
   System.out.println("lightYears:  " + lightYears + "
miles " + milesAway);
 }
```

Programming With Constants

```
public static void main (String[] args)
 {
    double lightYears, milesAway;
    final int LIGHTSPEED = 186000;
    final int SECONDS_PER_YEAR = 60*60*24*365;

    final double ALPHACENT_DIST = 4.35; // to AlphaCentauri
    final double ALDEBARAN_DIST = 68; // to Aldebaran

    lightYears = ALPHACENT_DIST;
    milesAway = lightYears * LIGHTSPEED * SECONDS_PER_YEAR;
    System.out.println("lightYears:  " + lightYears + "
miles " + milesAway);
    lightYears = ALDEBARAN_DIST;

    milesAway = lightYears * LIGHTSPEED * SECONDS_PER_YEAR;
    System.out.println("lightYears:  " + lightYears + "
miles " + milesAway);
 }
```

Avoiding Magic Numbers

- magic numbers: numeric constants directly in code
  - almost always bad idea!
    - hard to understand code
    - hard to make changes
    - typos possible
  - use constants instead

Questions?