Variables and Memory

What’s a variable, really?
What’s a type?

Not like variables in math

In math, \( x = x + 1 \) is just wrong
In Java, \( x = x + 1; \) has a meaning ...

Take the value of \( x \), add 1 to it, store the result in \( x \).

What’s a variable?

First try:
A variable is a name for a location in memory (a memory “cell”).
(not quite right ... but it’s a start)

When I write “\( x = 32 \)” I mean “put the value 32 in the memory location named ‘\( x \)’.”

What’s memory, really?

Memory cells are one big list, numbered from zero. The computer access them by “address” (number).

Variable ‘\( x \)’ might mean cell #6. \( x = 33 \) might mean:
put 100001 in cell #6
Memory

An address (location) is sent on some of these connectors

Contents go in or out on some of these connectors

01100010₂

Means 62₁₆ (interpreted as an integer)
Or 98₁₀
Or ‘b’ (interpreted as an ASCII character)
Or BOUND (interpreted as an x86 instruction)

Or ... it doesn’t “mean” anything, but we can interpret it several ways, as data or as a program instruction.

Memory Representation

It’s all binary (1’s and 0’s)

What does 01100010₂ mean?

(Trick question ... why?)

What’s a variable (2nd try)

A variable is a named location in memory, with an interpretation for the contents of that memory.

int x;

“x” is the name, “int” is how I interpret the bit patterns. Now I know that 01100010₂ means 98₁₀.
Dynamic types

Some languages (like PHP, Python, Perl, ...) store the type of the variable with the contents, in a “tag”. They don’t require variables to have declared types.

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>010110101110</td>
</tr>
<tr>
<td>1</td>
<td>010101101110</td>
</tr>
<tr>
<td>2</td>
<td>010110100000</td>
</tr>
<tr>
<td>3</td>
<td>110110101110</td>
</tr>
<tr>
<td>4</td>
<td>010110111110</td>
</tr>
<tr>
<td>5</td>
<td>011010101110</td>
</tr>
<tr>
<td>6</td>
<td>011110101110</td>
</tr>
<tr>
<td>7</td>
<td>010111101110</td>
</tr>
<tr>
<td>...</td>
<td>....</td>
</tr>
</tbody>
</table>

Tag: it's an integer
And this is its value

Static types in Java

int x;

No need to look at a tag – we will always interpret the value of x as an integer, and will never store anything else in the location of x.

Java, C, C++, C#, and many other languages do it this way. They have “static” or “compile-time” types.

Some Java types

<table>
<thead>
<tr>
<th>name</th>
<th>meaning</th>
<th>example (literal)</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>integer (represented as 32 bits)</td>
<td>42</td>
</tr>
<tr>
<td>long</td>
<td>integer (represented as 64 bits)</td>
<td>42L</td>
</tr>
<tr>
<td>float</td>
<td>floating point (approximation of real number), 32 bits</td>
<td>42.0f</td>
</tr>
<tr>
<td>double</td>
<td>floating point (approximation of real number), 64 bits</td>
<td>42.0</td>
</tr>
<tr>
<td>String</td>
<td>sequence of characters (text)</td>
<td>“Hello World”</td>
</tr>
<tr>
<td>char</td>
<td>one text character</td>
<td>‘h’</td>
</tr>
</tbody>
</table>

Operations depend on types!

15 + 32 is 47

“15” + “32” is “1532”
What is it?

int x = 15;
int y = 2;
int z;

z = x / y;

What value is in z?

What is it?

double x = 15.0;
double y = 2.0;
double z;

z = x / y;

What value is in z?

What is it?

int x = 15;
int y = 2;
int z;

z = x % y;

What value is in z?

What about these?

int x = 2.0;
double y = 3.53;
float z = 1.83f;
String s = “hello world”;
z = z / y;
y = y % z;
No no no no no nooooooo!

int x = 2.0;  // 2.0 is not an integer
double y = 3.53;  // ok
float z = 1.83;  // Should be 1.83f
String s = “hello world”;
z = z / y;  // Warning: precision
s = z;  // Incompatible types

So what’s with this?

int i = 42;  // life, the universe, and everything
float f = 42.0f;  // everything else
...
System.out.println(“The answer is ” + i + 
“or, if you prefer, “ + f);

How are we getting away with adding strings, integers, and floating point numbers?

Coercion

“Coercions” are implicit conversions from one type to another

“The answer is ” + 32
32 is converted to the string “32”

OK in printing. Otherwise, usually too confusing.

Casts

Like coercions, but explicit ...
float x = 42.84;
int y;
y = (int) x;  // Convert the value of x to integer
// and then store in y
System.out.println(y);

What does it print?
“Scope” of a variable

```java
{ ... ...
    int x;
    x = 3;
    ...
} x = 4;
```

The scope of x (up to the end of the current block)

Wrong! x is out of scope.

Why do variables have limited scope?

Suppose you are designing a new and better programming language ... Java+++

Will variables in your language follow scope rules? Why or why not?

Some reasons for scope

I can (safely) reuse variable names, (so my names can be shorter)

I know what I can forget

I can get more error messages from the compiler (yes, really that’s a good thing)

Scoping variables

Narrower is better

- Declare variables in as small a scope as practical

```java
static void foo() {
    int x;
    x = 14;
    y = x / 2;
    ... }```

Wider scope

```java
    ...}
    int x;
    { ... } Narrower scope
```
Variable names matter

The computer doesn’t care ... but other programmers (and graders!) do

“dollars” and “cents” are better than “x” and “y”

but_this_is_not_really_a_very_good_name longer ≠ better

---

Naming conventions in Java

ClassNames use CamelCase
variableNames use camelCase too but they start with a lower case letter
CONSTANTS_SHOUT (especially if they have wide scope)

The wider the scope, the more descriptive a variable name needs to be

---

Summary: What’s a variable?

It names a location in memory
It has a type, which determines how the bits in memory are interpreted, and what operations we can apply to the value
It has a scope (the narrower the better)
Its name should be descriptive (enough) and follow Java naming conventions