CS1007 lecture #4 notes

thu 12 sep 2002

- news
- data types and storage
- variables and assignment
- binary numbers and arithmetic
- ASCII
- Strings
- math operators
- increment and decrement operators
- reading: *ch 2.5,2.7-2.12*

news.

- always check class web: *http://www.columbia.edu/˜cs1007*
- homework #1 is due next tuesday
- recitation lists are being posted on the web...
- if you haven't signed up yet, email Min Co (mtc38@columbia.edu) and the recitation TA
- TAs will also have office hours in the TA room in Mudd (also posted on the web)
- check out the "human help" link for all office and recitation hours

data and storage.

- last week we talked about output
- programs = objects + methods
- objects = data
- data must be *stored*
- all storage is numeric (0's and 1's)

memory.

- think of the computer's memory as a bunch of boxes
- inside each box, there is a number
- you give each box a name
  ⇒ defining a *variable*
- example:

program code:	*computer's memory:*
`int x;`	x → [ ]

variables.

- variables have:
  - name
  - type
  - value
- naming rules:
  - names may contain letters and/or numbers
  - but cannot begin with a number
  - names may also contain underscore (_) and dollar sign ($)
  - underscore is used frequently; dollar sign is not too common in Java
  - can be of any length
  - cannot use Java keywords
  - Java is *case-sensitive!!*

primitive data types.

- numeric

byte	8 bits	-128 = -2^7	127 = 2^7 - 1
short	16 bits	-32,768 = -2^{15}	32,767 = 2^{15} - 1
int	32 bits	-2^{31}	2^{31} - 1
long	64 bits	-2^{63}	2^{63} - 1
float	32 bits	≈ -3.4E+38, 7 sig dig	≈ 3.4E+38, 7 sig dig
double	64 bits	≈ -1.7E+308, 15 sig dig	≈ 1.7E+308, 15 sig dig

- boolean

boolean	1 bit

- character

char	16 bits

- 7 bits for ASCII
- 8 bits for extended ASCII
- 16 bits for Unicode

assignment.

- = is the assignment operator
- example:

program code:

```
int x; // declaration
x = 19; // assignment
```

or

```
int x = 19;
```

computer's memory:

x → [19]

storage is binary.

x → [19]

is really stored like this:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1

this is base 2!

$19_{10} = 10011_2$

remember bases?

base 10:

$362 = (2 * 1) + (6 * 10) + (3 * 100)$

$= (2 * 10^0) + (6 * 10^1) + (3 * 10^2)$

base 2:

$1 = 2^0 = 1$

$10 = 2^1 = 2$

$100 = 2^2 = 4$

$1000 = 2^3 = 8$

$10000 = 2^4 = 16$

...

so

$10011_2 = (1 * 2^0) + (1 * 2^1) + (0 * 2^2) + (0 * 2^3) + (1 * 2^4)$

$= (1 * 1) + (1 * 2) + (0 * 4) + (0 * 8) + (1 * 16)$

$= 19_{10}$

base conversion: 2 to 10.

$1010100_2 =$

	=	=
$(0 * 2^0)$	$(0 * 1)$	0
$+ (0 * 2^1)$	$+ (0 * 2)$	+ 0
$+ (1 * 2^2)$	$+ (1 * 4)$	+ 4
$+ (0 * 2^3)$	$+ (0 * 8)$	+ 0
$+ (1 * 2^4)$	$+ (1 * 16)$	+ 16
$+ (0 * 2^5)$	$+ (0 * 32)$	+ 0
$+ (1 * 2^6)$	$+ (1 * 64)$	+ 64

$= 84_{10}$

base conversion: 10 to 2.

$84_{10} =$

84 / 2	=	42	rem 0
42 / 2	=	21	rem 0
21 / 2	=	10	rem 1
10 / 2	=	5	rem 0
5 / 2	=	2	rem 1
2 / 2	=	1	rem 0
1 / 2	=	0	rem 1

$\Rightarrow 1010100_2$

two tricks.

base 8 (octal):

000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

base 16 (hexadecimal, “hex”):

0000	0	1000	8
0001	1	1001	9
0010	2	1010	A (10)
0011	3	1011	B (11)
0100	4	1100	C (12)
0101	5	1101	D (13)
0110	6	1110	E (14)
0111	7	1111	F (15)

- replace each octal (or hex) digit with the 3 (or 4) digit binary
- replace every 3 (or 4) binary digits with one octal (or hex) digit

back to storage.

x → 19

is really stored like this:

31	30	...	7	6	5	4	3	2	1	0
0	0	...	0	0	0	1	0	0	1	1

- bits are numbered, from right to left, starting with 0
- highest (rightmost, "most significant") bit is *sign* bit

ASCII.

- ASCII = American Standard Code for Information Interchange
- characters are stored as numbers
- standard table defines 128 characters
- example:

```
char c = 'A';
```

'A' = 65_{10} = 01000001_2

c →

7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	1

Strings.

- a `String` in Java is a special data type — it's called a *wrapper class* (which we'll talk about in detail later)
- a `String` is essentially a group of `char`s
- it comes with a *method* called `length()` that lets you find out how many characters are in the string (i.e., how long it is)
- it comes with a number of other methods, which we'll talk about later
- a `char` has single quotes around it

  ```
  char c = 'A';
  ```

- a `String` has double quotes around it

  ```
  String s = "hello world!";
  ```

- in this case, the method `s.length()` returns 12

mathematical operators.

+	unary plus
−	unary minus
+	addition
−	subtraction
∗	multiplication
/	division
%	modulo

example:

```
int x, y;
x = -5;
y = x * 7;
y = y + 3;
x = x * -2;
y = x / 19;
```

what are `x` and `y` equal to?

modulo means "remainder after integer division"

coercion or type casting.

- remember from last time: data of type `char` is stored as a number — which is really an index into the ASCII table
- a declaration like this:

```
char y = 'A';
```

really stores a 65 (the ASCII value of 'A') in a memory location that is labeled `y`

- you can do math on that 65 by *coercing* (aka *type casting*) the `char` to an `int`
- for example:

```
char y = 'A';    // initialize variable y to store an A
int  x = (int)y; // initialize variable x to store 65
x = x + 1;       // increment x (to 66)
y = (char)x;     // coerce x from an int to a char ('B')
```


increment and decrement operators.

- increment: ++
  `i++;`
  is the same as:
  `i = i + 1;`
- decrement: −−
  `i--;`
  is the same as:
  `i = i - 1;`

assignment operators.

`+=`
`i += 3;` is the same as: `i = i + 3;`

`-=`
`i -= 3;` is the same as: `i = i - 3;`

`*=`
`i *= 3;` is the same as: `i = i * 3;`

`/=`
`i /= 3;` is the same as: `i = i / 3;`

`%=`
`i %= 3;` is the same as: `i = i % 3;`