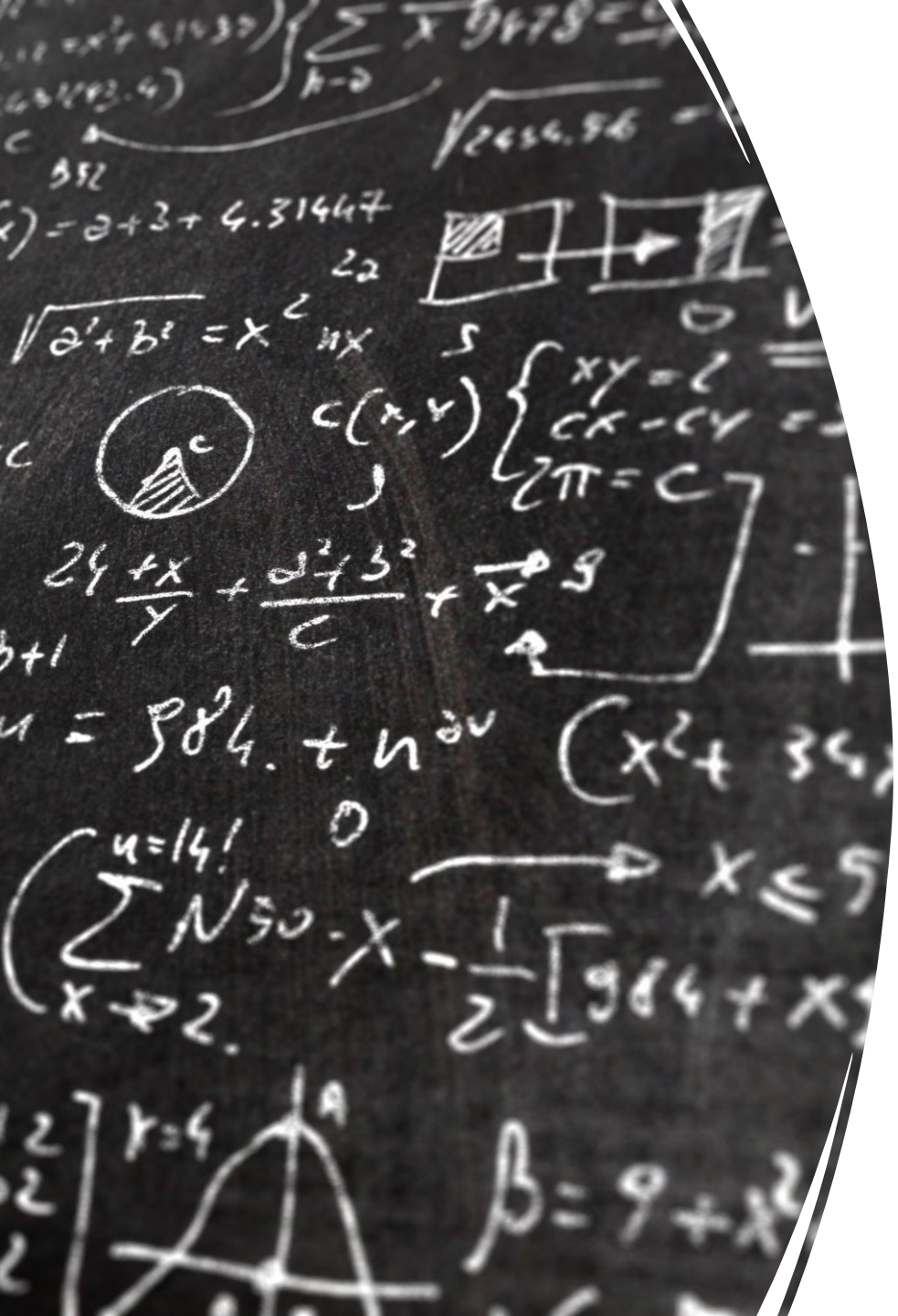


CS 113 – Computer Science I

Lecture 8 – Arrays, Recursion

Adam Poliak

02/09/2023

Announcements

- Assignment 02 – moved deadline for tomorrow
- Assignment 03 – released last night
  - Due Wednesday 02/15
- Today’s office hours:
  - 3:00 – 4:45

Agenda

- Announcements
- Recap
- Arrays
- Recursion

Style

- How we format our programs is **very** important
  - Like rules of etiquette around eating and keep a clean appearance
  - Like punctuation rules, it helps make text more readable
- Variable names should be descriptive
- Indentation is **very** important
  - Every statement inside a pair of braces must be indented
- Braces should be placed consistently

Comparing strings

- In Java, you cannot directly compare strings: use **compareTo**

```
String a = “apple”;
String b = “banana”;
if (a.compareTo(b) == 0) {
   System.out.println(“a and b match!”);
}
if (a.compareTo(b) != 0) {
 System.out.println(“a and b DO NOT match!”);
}
```

Lexicographic Values/Order

- Strings are **ordered lexicographically**
  - Generally, the same order as alphabetical order, with some caveats
  - The characters of a string each correspond to a number

ASCII

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

https://www.asciitable.com/

StringCompare.java

```
String first = "a";
String second = "A";
int asciia = (int) first.charAt(0);
int asciib = (int) second.charAt(0);
System.out.println("ASCII Code for "+first+" is " + asciia);
System.out.println("ASCII Code for "+second+" is " + asciib);

if (first.compareTo(second) == 0) {
   System.out.println(first+" is equal to "+second);
}
else if (first.compareTo(second) < 0) {
   System.out.println(first+" is less than "+second);
}
else if (first.compareTo(second) > 0) {
   System.out.println(first+" is greater than "+second);
}
```

```
$ java StringCompare
ASCII Code for a is 97
ASCII Code for A is 65
a is greater than A
```

Exercise: IsPrimary

Write a program that asks the user for a color and prints whether the color is primary or not.

- The primary colors are “red”, “green”, “blue”
- All other inputs are non-primary

```
$ java IsPrimary
Enter a color: green
green is not primary

$ java IsPrimary
Enter a color: blue
blue is primary
```

Agenda

- Announcements
- Recap
- **Arrays**
- Recursion

Arrays

Arrays

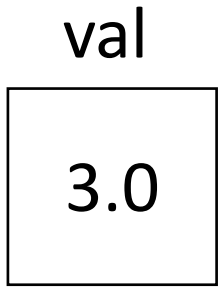
Idea: Store multiple values into a single variable

Values are sequential

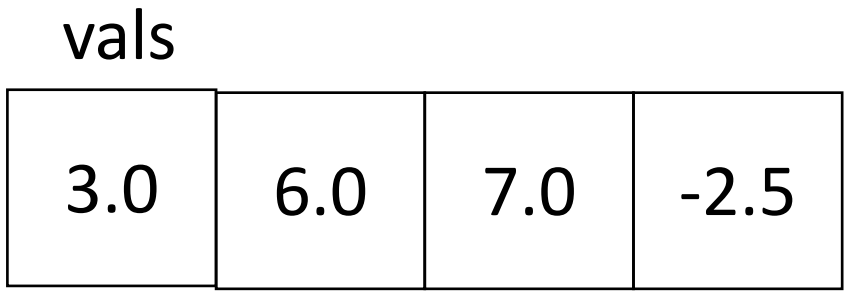
Analogous to a list

Arrays

```
double val = 3.0;
```




```
double[] vals = {3.0, 6.0, 7.0, -2.5};
```



Arrays

```
boolean[] flags = {true, false};
```

```
String[] greetings = {"hi", "hola", "ciao", "aloha"};
```

Arrays

Three ways to initialize an array

1. With an initial value

2. With allocated space, but uninitialized

3. With an empty array reference

Arrays

Three ways to initialize an array

1. With an initial value

```
int[] numbers = {1, 2, 5};
```

2. With allocated space, but uninitialized

```
int[] numbers = new int[3];
```

3. With an empty array reference

```
int[] numbers = null;
```

Array Indexing

Access individual elements of an array with indexing

array[index]

We use *zero*-based indexing

first element is **0**

last element is **length-1**

Accessing indices out of range results in a **runtime error**!

Exercise: print backwards

Write a program, Backwards.java, that asks the user for 5 integers and then prints the list of numbers in reverse order

Strings

Strings are implemented as *arrays of characters*

Get the length of a string with length()

```
String greeting = “hola”;
int len = greeting.length(); // what is the length?
char c = greeting[2]; // what character is in index 2?
```

char: New built-in type, denoted with single quote, e.g. ‘a’ or ‘{‘

Exercise: GetCharacters.java

Write a program, GetCharacters.java, that asks the user for a word and then prints the first, last and middle character.

```
Enter a word: hola!
FirstIndex: 0 FirstCharacter: h
MiddleIndex: 2 MiddleCharacter: l
LastIndex: 5 LastCharacter: !
```

Command line arguments

```
public static void main(String[] args)
```

Command line arguments are an *array of String*

Exercise: Write a program called commandLineArgs.java that prints out 3 command line arguments that are passed in.

Agenda

- Announcements
- Recap
- Arrays
- **Recursion**

Washing dishes

Smart way to wash dishes

Punt the problem to someone else

But we want to wash one dish so we can say we washed a dish

Motivation #2 - adding

I’m going to give you a list of numbers

- Group A: each person adds up all the numbers
- Group B: one person takes the first number and passes the rest to the next person, repeat this process until no more numbers
  - Last person adds the last two numbers and send the result to the previous person
    - Who adds their number with the result …

Motivation #2 adding numbers

- 20,
- 43,
- 13,
- 13,
- 10,
- 43,
- 90,
- 32,
- 42

Motivation #2 adding numbers

- Which was easier?
- Which was like ”smarter” washing dishes?
  - How so?
- This is called recursion

Recursion

a function that calls itself

“Simple” way to solve “similar” problems

Creating a recursive algorithms

Rule that “does work” then ”calls itself” on a smaller version of the problem

Base case that handles the smallest problem

Prevents “infinite recursion”

Recursion example – print “hello” 5 times

Rule: Print “hello” once and then print “hello” 4 times

Base case: When the number of times to print is 0, stop printing

Recursive functions – base case

Conditional statement that prevents infinite repetitions

Usually handles cases where:

- input is empty
- problem is at its smallest size

Recursion Example - Factorial

$$n! = n * (n-1) * (n-2) * \ldots * 1$$

3! = 3 * 2 * 1 = 6

4! = 4 * 3 * 2 * 1 = 24

Visualizing recursion – Factorial example

factorial(5) =

= 5 * factorial(4)

= 5 * 4 * factorial(3)

= 5 * 4 * 3 * factorial(2)

= 5 * 4 * 3 * 2 * factorial(1)

= 5 * 4 * 3 * 2 * 1

Recursion Example – Contains letter

Recursion Visualization – Contains letter

contains(“l”, “apple”) =

contains(“l”, “apple”)

contains(“l”, “pple”)

contains(“l”, “ple”)

contains(“l”, “le”, 3)

return true

Recursion Example – printList

Write a recursive function that prints the contents of an array

Recursion limitations

- Limited number of times we can recurse
  - Stackoverflow – too many frames
- Potentially memory inefficient
  - If we copy data in subproblems – we'll worry about this in a few weeks
- Performance: might duplicate unnecessary work
  - We'll define performance later in the semester