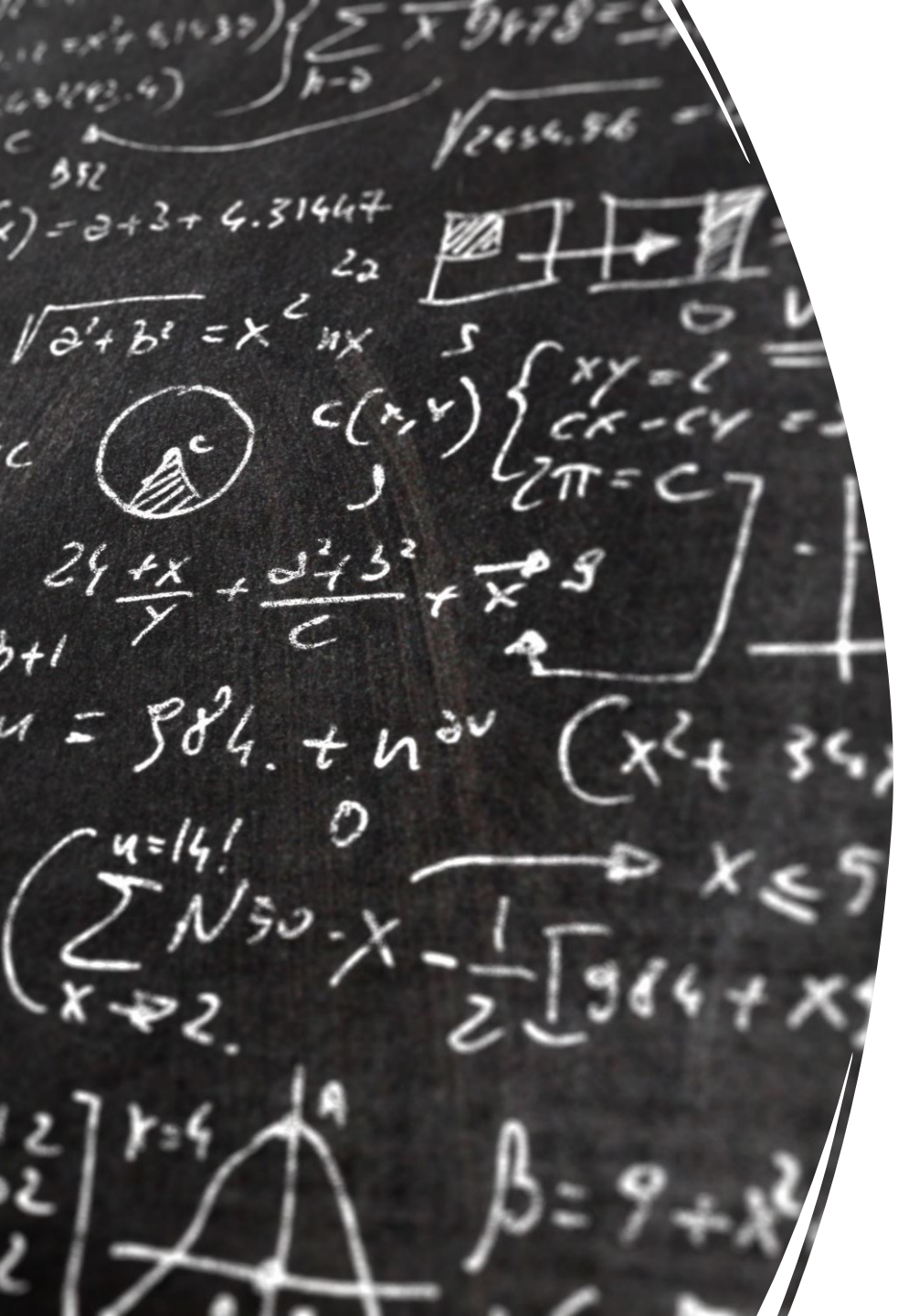


CS 113 – Computer Science I

Lecture 11 - Loops

Adam Poliak

02/22/2023

Announcements

- HW04 –
  - due tomorrow night (Wednesday 02/22)
- HW05 - loops
  - Due Monday 02/27
  - Short implementing just 7 methods
- Post spring break homeworks:
  - Due Friday nights

Midterm – Thursday 03/02

In class, closed book

variables (int, double, char, bool, string, array)
Expressions
Methods
Frame diagrams
Conditionals
Recursion
Loops

Agenda

- Announcements
- **While Loops**
- For Loops

Exercise

Suppose we wanted to ask the user for 6 numbers (int) and output their sum?

Loops

- Easy way to repeat some computation
- Two kinds of loops:
  - While
  - For
- Loops repeat block of code until the condition becomes false

Example: While Loop

```
int val = 0;
int sum = 0;

int count = 0;
while (count < 6) {
    System.out.print("Enter a number: ");
    val = sc.nextInt();
    sum = sum + val;
    count = count + 1;
}
System.out.println("The sum is "+sum);
```

Tracing Loops

```
int sum = 1;
int count = 0;
while (count < 3) {
    sum = sum + 2;
    count = count + 1;
}
```

Iteration	Count < 6	count	sum

Tracing Loops

```
int sum = 1;
int count = 0;
while (count < 3) {
    sum = sum + 2;
    count = count + 1;
}
```

Iteration	Count < 6	count	sum
0	T	0	1

Tracing Loops

```
int sum = 1;
int count = 0;
while (count < 3) {
     sum = sum + 2;
     count = count + 1;
}
```

Iteration	Count < 6	count	sum
0	T	0	1
1	T	1	3

Tracing Loops

```
int sum = 1;
int count = 0;
while (count < 3) {
    sum = sum + 2;
    count = count + 1;
}
```

Iteration	Count < 6	count	sum
0	T	0	1
1	T	1	3
2	T	2	5

Tracing Loops

```
int sum = 1;
int count = 0;
while (count < 3) {
    sum = sum + 2;
    count = count + 1;
}
```

Iteration	Count < 6	count	sum
0	T	0	1
1	T	1	3
2	T	2	5
3	T	3	7

Exercise: Tracing loops

```
int sum = 10;
int count = 0;
while (count < 6) {
    sum = sum - 1;
    count = count + 2;
}
```

Iteration	Count < 6	count	sum

Accumulator pattern

Idea: Repeatedly update a variable (typically in a loop)

Pattern:

1. Initialize accumulator variable
2. Loop until done
    1. Update the accumulator variable

Convenience syntax: Assignment

Because updating variable values is so common, language such as Java provide shorthand syntax for it

- Analogy: contractions in English

sum = sum + 2

count = count + 1

count = count – 1

product = product * 2

divisor = divisor / 2

message = message + “lol!”

Convenience syntax: Assignment

Because updating variable values is so common, language such as Java provide shorthand syntax for it

- Analogy: contractions in English

sum = sum + 2	
count = count + 1	
count = count - 1	
product = product * 2	
divisor = divisor / 2	
message = message + “ lol”	

Convenience syntax: Assignment

Because updating variable values is so common, language such as Java provide shorthand syntax for it

- Analogy: contractions in English

sum = sum + 2	sum += 2
count = count + 1	
count = count - 1	
product = product * 2	
divisor = divisor / 2	
message = message + “ lol”	

Convenience syntax: Assignment

Because updating variable values is so common, language such as Java provide shorthand syntax for it

- Analogy: contractions in English

sum = sum + 2	sum += 2
count = count + 1	count += 1
count = count - 1	
product = product * 2	
divisor = divisor / 2	
message = message + “ lol”	

Convenience syntax: Assignment

Because updating variable values is so common, language such as Java provide shorthand syntax for it

- Analogy: contractions in English

sum = sum + 2	sum += 2
count = count + 1	count += 1
count = count - 1	count -= 1
product = product * 2	
divisor = divisor / 2	
message = message + “ lol”	

Convenience syntax: Assignment

Because updating variable values is so common, language such as Java provide shorthand syntax for it

- Analogy: contractions in English

sum = sum + 2	sum += 2
count = count + 1	count += 1
count = count - 1	count -= 1
product = product * 2	product *= 2
divisor = divisor / 2	
message = message + “ lol”	

Convenience syntax: Assignment

Because updating variable values is so common, language such as Java provide shorthand syntax for it

- Analogy: contractions in English

sum = sum + 2	sum += 2
count = count + 1	count += 1
count = count - 1	count -= 1
product = product * 2	product *= 2
divisor = divisor / 2	divisor /= 2
message = message + “ lol”	

Convenience syntax: Assignment

Because updating variable values is so common, language such as Java provide shorthand syntax for it

- Analogy: contractions in English

sum = sum + 2	sum += 2
count = count + 1	count += 1
count = count - 1	count -= 1
product = product * 2	product *= 2
divisor = divisor / 2	divisor /= 2
message = message + “ lol”	message += “ lol”

Exercise: Write a program that computes powers of 2

Write a program, LoopPow2.java, that computes powers of twos. For example,

```
$ java LoopPow2
Enter an exponent: 0
2 to the power of 0 is 1

$ java LoopPow
Enter an exponent: 1
2 to the power of 1 is 2

$ java LoopPow
Enter an exponent: 4
2 to the power of 4 is 16
```

Agenda

- Announcements
- While Loops
- **For Loops**

Example: For Loop

```
int val = 0;
String valStr = "";
int sum = 0;

for (int count = 0; count < 6; count = count +1) {
    System.out.print("Enter a number: ");
    valStr = System.console().readLine();
    val = Integer.parseInt(valStr);
    sum = sum + val;
}
System.out.println("The sum is "+sum);
```

Example: For Loop

initialize

condition

update

```
for (int count = 0; count < 6; count = count +1) {

}
```

Exercise: Tracing loops

```
String pattern = "";
for (int i = 0; i < 3; i++) {
    pattern = pattern + "*";
}
System.out.println(pattern);
```

Iteration	i < 3	i	pattern

Exercise: Tracing loops

```
String pattern = "";
for (int i = 0; i < 3; i++) {
    pattern = pattern + "*";
}
System.out.println(pattern);
```

Iteration	i < 3	i	pattern
0	T	0	""
1	T	1	"*"
2	T	2	"**"
3	F	3	"***"

Exercise: LoopPattern.java

```
$ java LoopPattern
Enter a length: 5
*-*-*

$ java LoopPattern
Enter a length: 10
*-*-*-*-*-

$ java LoopPattern
Enter a length: 0


$ java LoopPattern
Enter a length: 1
*
```

Exercise: Nested loops

```
$ java Square
Enter a size: 5
*****
*****
*****
*****
*****

$ java Square
Enter a size: 1
*

$ java Square
Enter a size: 0
```

Iterating through an array

Write a method called printArray that takes in an array of integers and prints out the values in each array:

printArray({1,2,3,4}) -> “1 2 3 4”

Bank example

Keep track of account balances

Use an array:

- Each index represents another account
- The value represents the account's balance

Determine how many accounts we can hold:

- Create a new array of fixed size

Bank example

Over time our bank becomes successful, lots of new clients

No more space for new customers

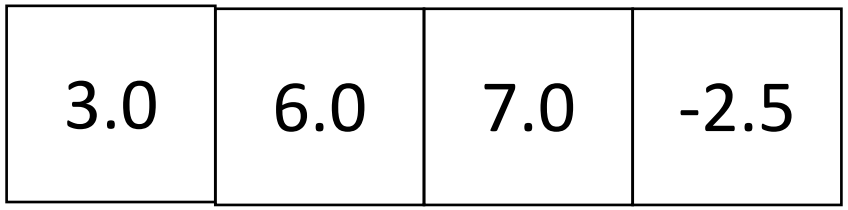
Implementation issue: running out of space in our array

Solution: build a bigger bank!

Building a bigger bank

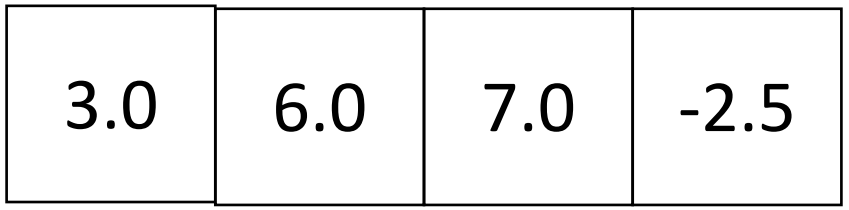
Copying arrays

Old bank

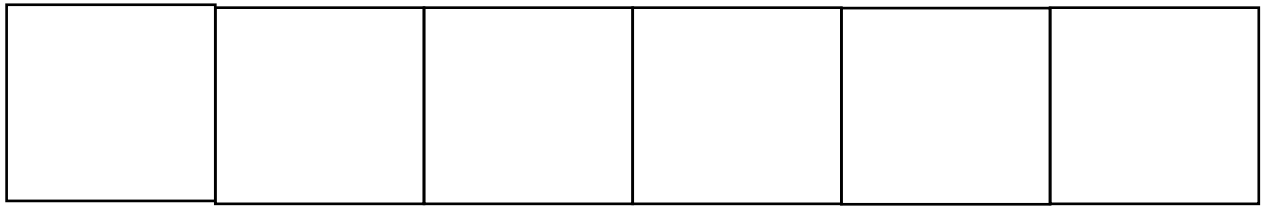


Copying arrays – build the new bank/array

Old bank

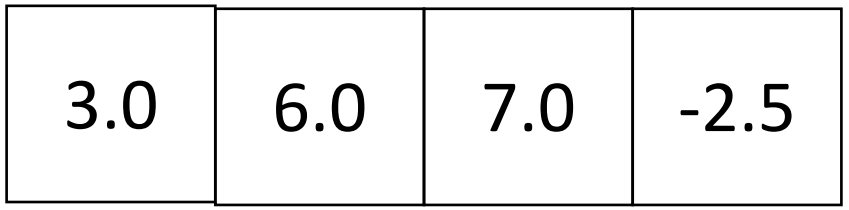


new bank

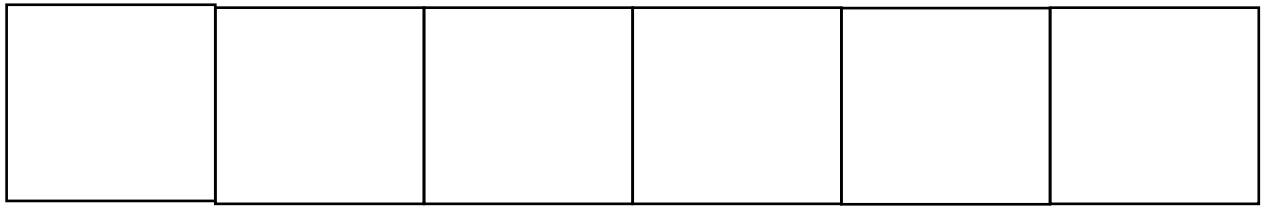


Copying arrays – copy over values/customers

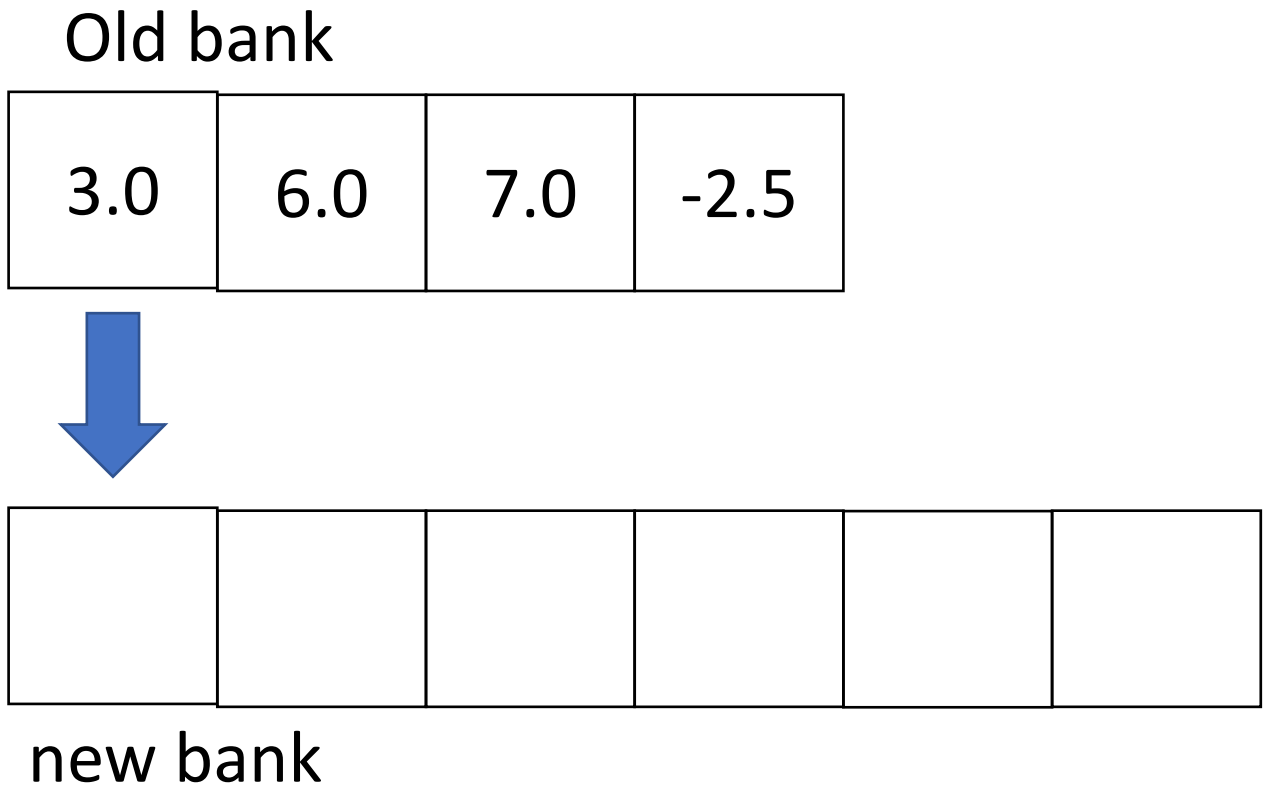
Old bank



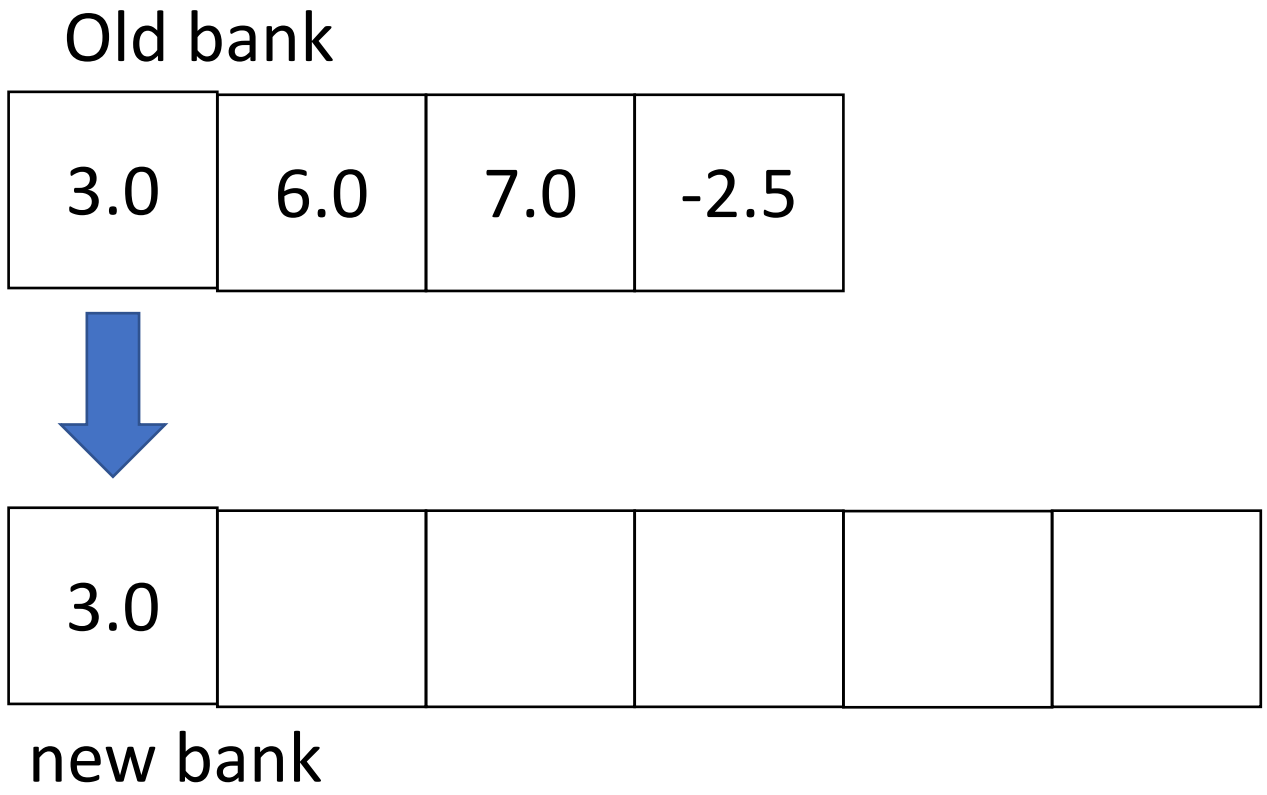
new bank



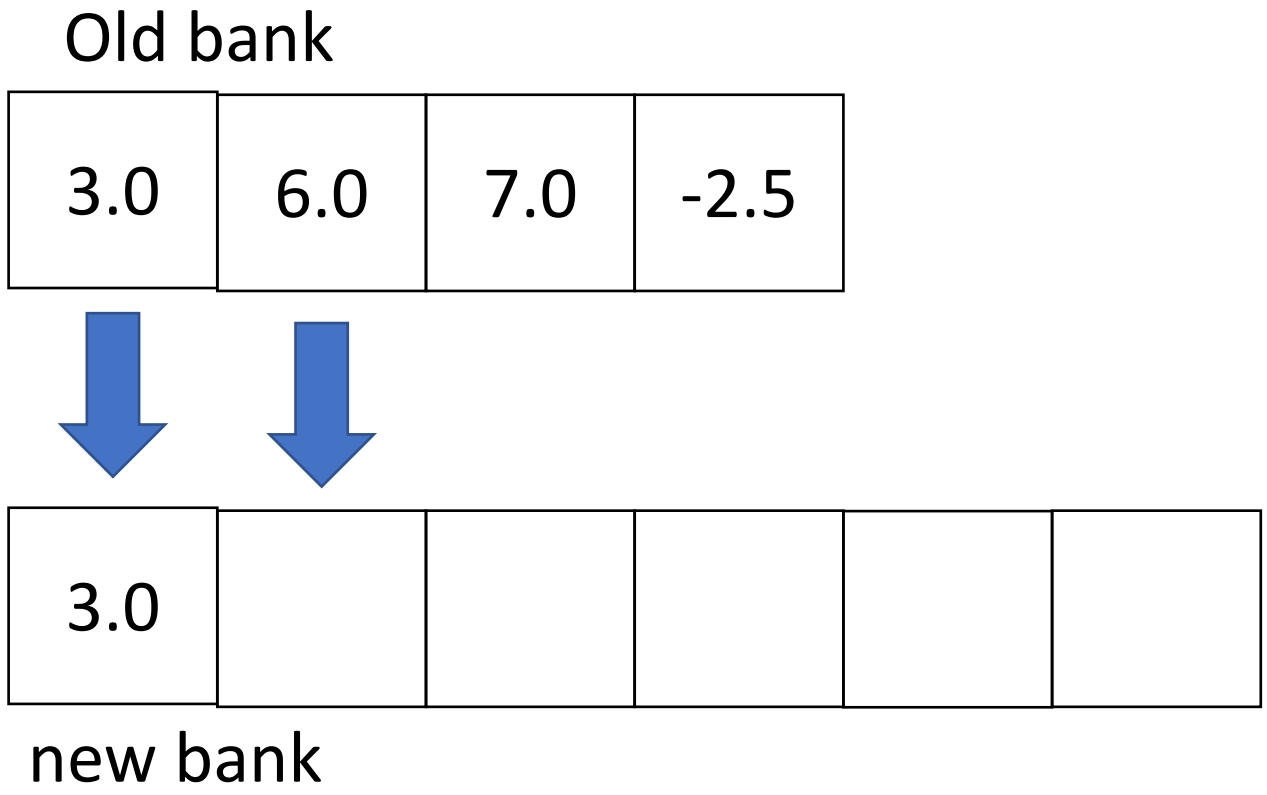
Copying arrays – copy over values/customers



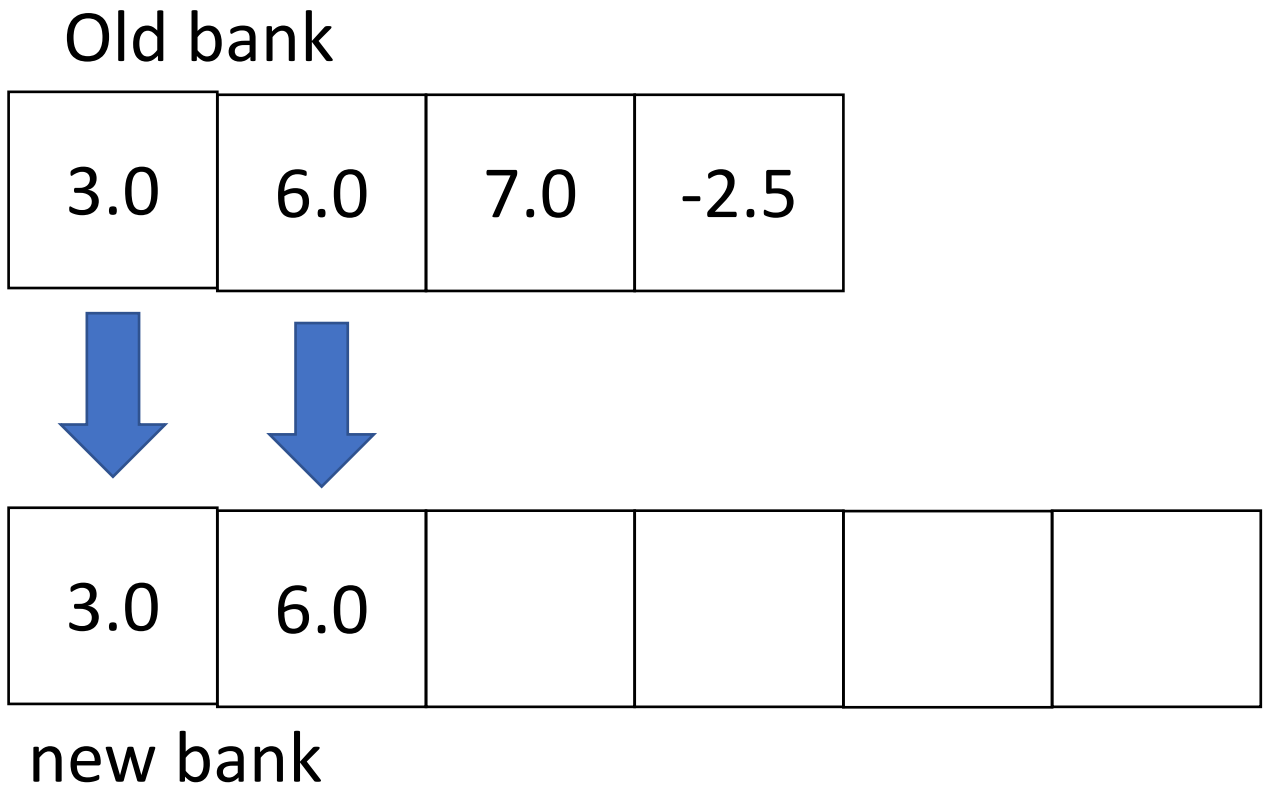
Copying arrays – copy over values/customers



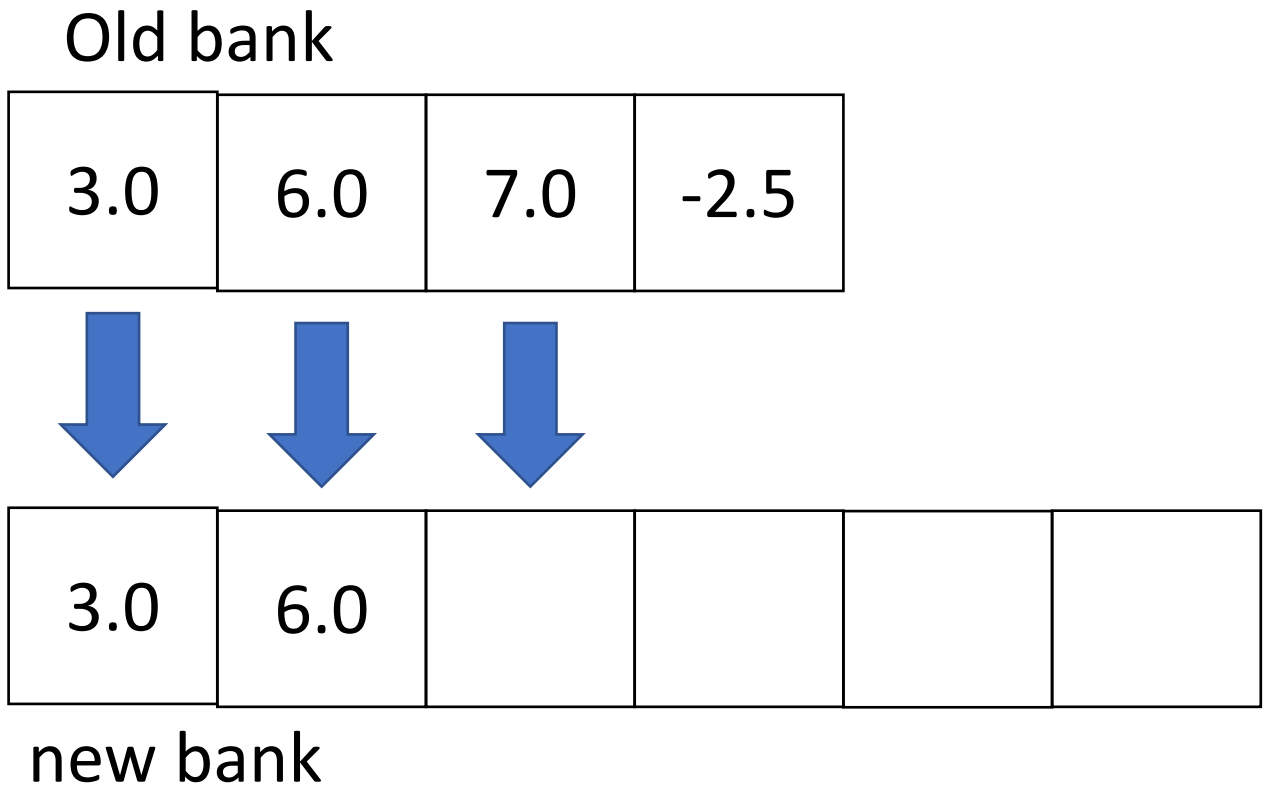
Copying arrays – copy over values/customers



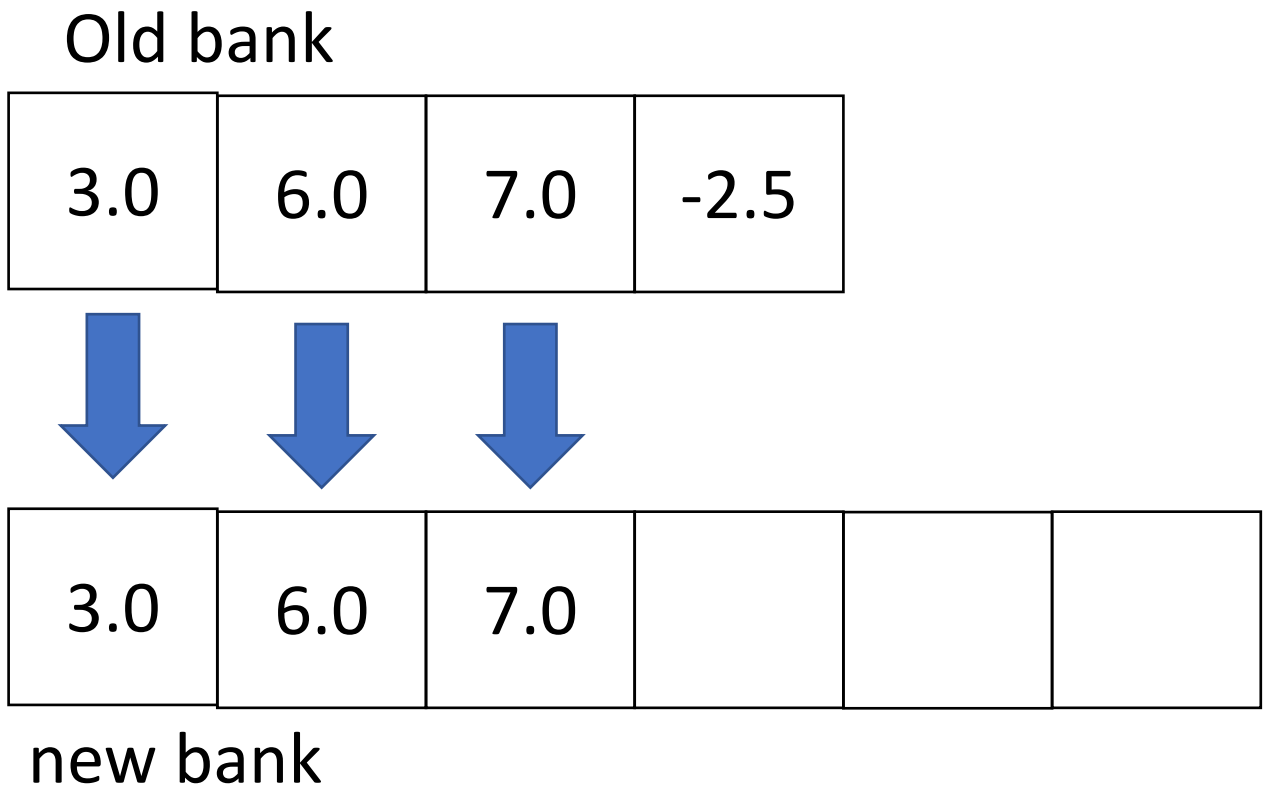
Copying arrays – copy over values/customers



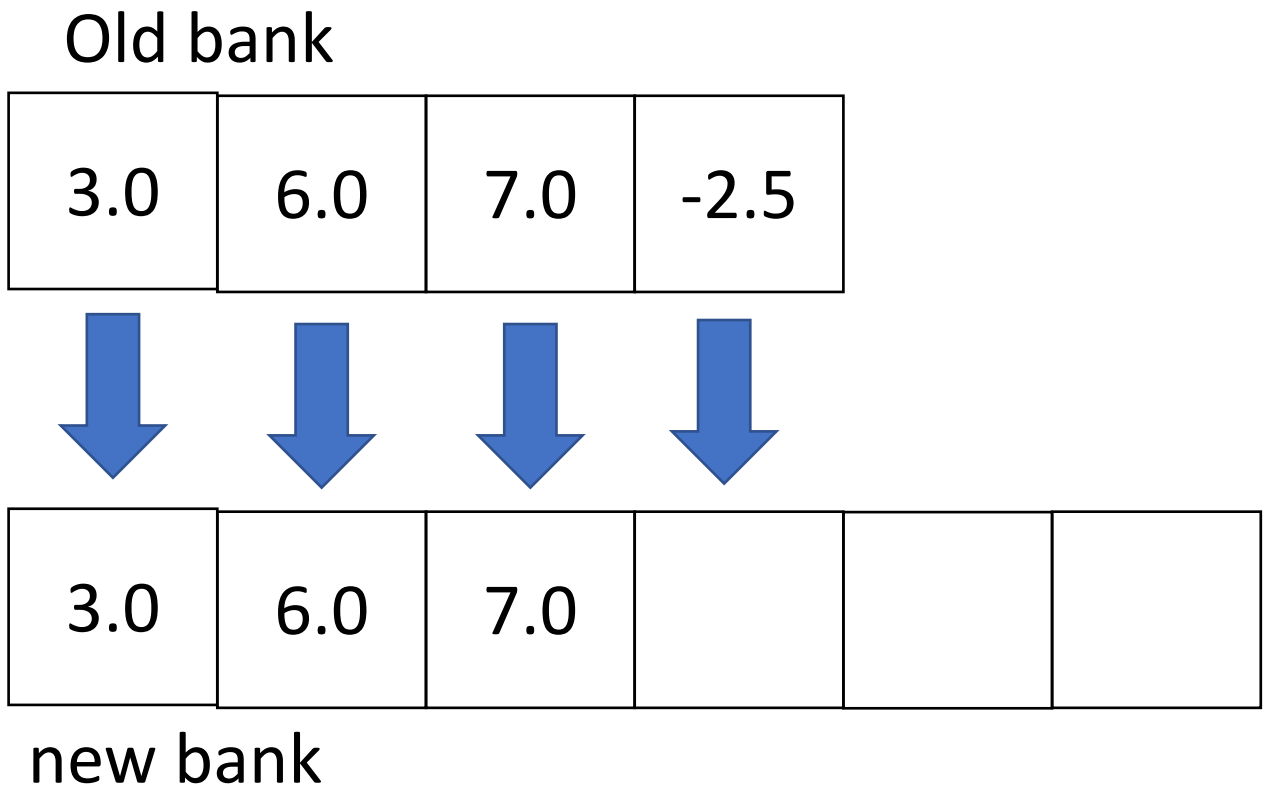
Copying arrays – copy over values/customers



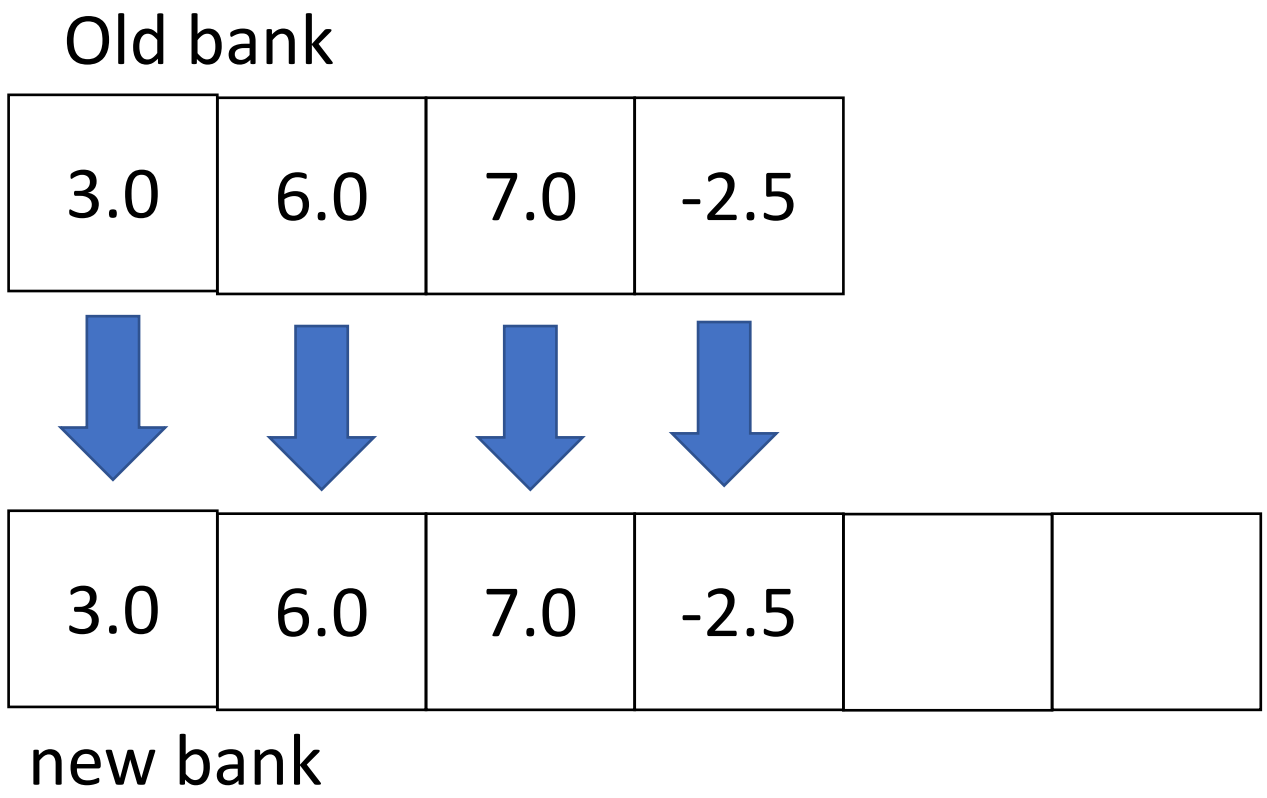
Copying arrays – copy over values/customers



Copying arrays – copy over values/customers



Copying arrays – copy over values/customers



Algorithm

When we run out of space in an array

- Create a new array (that’s a bit bigger)
- Copy over all elements from the older array to the new array

How big should the new array be?

Previous size plus 1

- Pro: not making too much space
- Con: might have to create new arrays a lot of times

As big as possible

- Pro: rarely have to create a new array
- Con: wasted space

Typical solution – previous size x 2