COMPSCI 121: RECURSION

SPRING 2020

GOALS FOR TODAY'S CLASS

Introduction to Recursion and its implementation:

- example with `int`
- example with Strings

Finding the right way to decompose a complex problem into manageable sub-problems is an important problem-solving strategy.

Recursion is an elegant way to decompose and solve some complex problems.

ITERATIVE SOLUTION

Task: compute the sum of the first n numbers.
Example: n=4 sum = 1 + 2 + 3 + 4 = 10.
Assume n>0.

A looping (iterative) approach:

```
int sum = 0;
for (int i = 1; i <= n; i++)
    sum+=i;
System.out.print(sum);
```

Use the variables `i` and `sum` to do the counting and keep the running sum .

RECURSIVE SOLUTION

Task: compute the sum of the first n numbers.

Example: n=4 sum = 1 + 2 + 3 + 4 = 10. Assume n>0.

A recursive approach:

sum of first 4 numbers = 4 + sum of first 3 numbers

sum of first 3 numbers = 3 + sum of first 2 numbers

sum of first 2 numbers = 2 + sum of first 1 numbers

sum of first 1 numbers = 1

RECURSIVE SOLUTION

```
Task: compute the sum of the first n numbers.
Example: n=4   sum = 1 + 2 + 3 + 4 = 10. Assume n>0.
```

A recursive approach:

simpler versions of the problem

```
sum of first 4 numbers = 4 + sum of first 3 numbers
sum of first 3 numbers = 3 + sum of first 2 numbers
sum of first 2 numbers = 2 + sum of first 1 numbers
sum of first 1 numbers = 1
```

base case

Recursion needs two things: 1- Base case, when the task is done, 2- decomposition: find a *simpler version* of the problem to solve and combine them to solve the big problem.

RECURSIVE SOLUTION

```
Task: compute the sum of the first n numbers.
Example: n=4   sum = 1 + 2 + 3 + 4 = 10. Assume n>0.
```

A recursive approach:

simpler versions of the problem

```
sum of first 4 numbers = 4 + 6
sum of first 3 numbers = 3 + 3
sum of first 2 numbers = 2 + 1
sum of first 1 numbers = 1
```

base case

Recursion needs two things: 1- Base case, when the task is done, 2- decomposition: find a *simpler version* of the problem to solve and combine them to solve the big problem.

RECURSIVE SOLUTION

```
Task: compute the sum of the first n numbers.
Example: n=4   sum = 1 + 2 + 3 + 4 = 10.
Assume n>0.

Pseudocode:
   if n is 1
     return 1;
   else
     return n + sum of first n-1 numbers
```

check for the base case

combining solutions to simpler versions of the problem

Need to know that the base case will be reached. Each recursive step is on n-1, so for any value of n we will eventaully reach the base case of n=1.

RECURSIVE METHOD JAVA IMPLEMENTATION

```
public static int sum(int n)  {
      if(n == 1)
         return 1;
      else
         return n + sum(n-1);
}
```

Base case

Recursive call to smaller sub-problems.

What do you think this method does if `n = 5`?

RECURSIVE METHOD CALLS

```
Task: compute the sum of the first n numbers.
Example: n=5   sum = 1 + 2 + 3 + 4 + 5 = 15.
Tracing the steps of a recursive method:
    sum(5)
    5 + sum(4)
        4 + sum(3)
            3 + sum(2)
                2 + sum(1)
                    1
```

Recursive calls to `sum`.

Base case: `sum` returns 1

"Progress" is made by passing a simpler problem to `sum` until the base case is reached.

TRACING A RECURSIVE METHOD

How does the method work?

```
Enter sum 5
  Enter sum 4
    Enter sum 3
      Enter sum 2
        Enter sum 1
        Return 1
      Return 3
    Return 6
  Return 10
Return 15
Sum of 1 to 5 is:15
```

Calls to `sum(n-1)` being made.

Return path

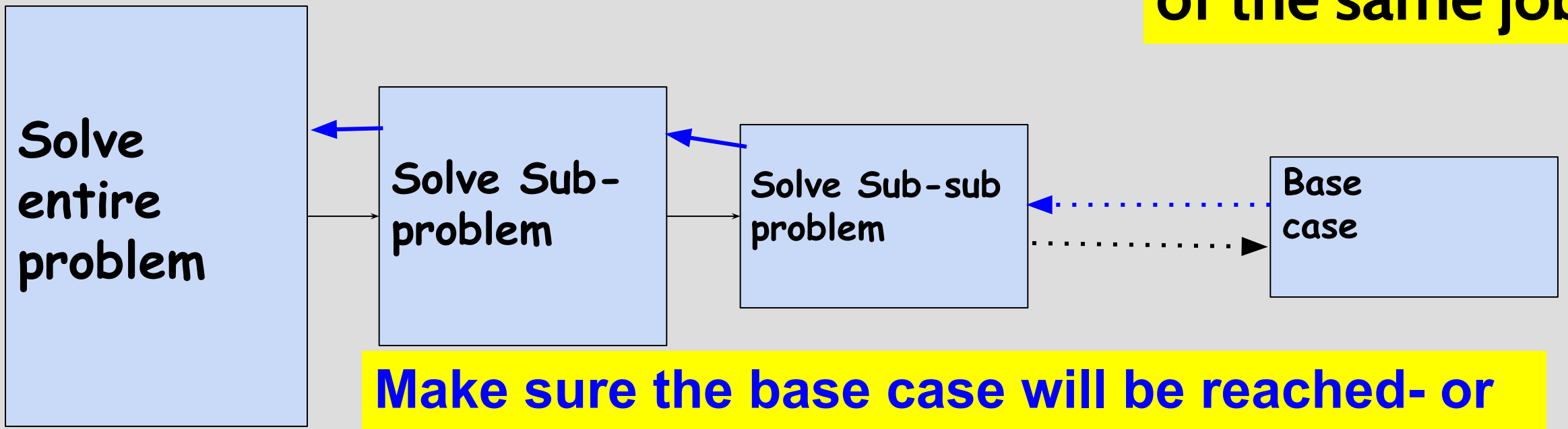
```
public static int sum(int n)
    {
      if(n == 1)
        return 1;
      else
        return n + sum(n-1);
     }
```

Each call "defers" execution to the next call. The addition is done by combining the returns.

RECURSIVE METHODS- IN GENERAL

```
void recursiveMethod(arg){
   … …
   recursiveMethod(smaller arg);
}
```

Method calls itself (to do a smaller version) of the same job.



Make sure the base case will be reached- or it never stops! Equivalent of endless loop.

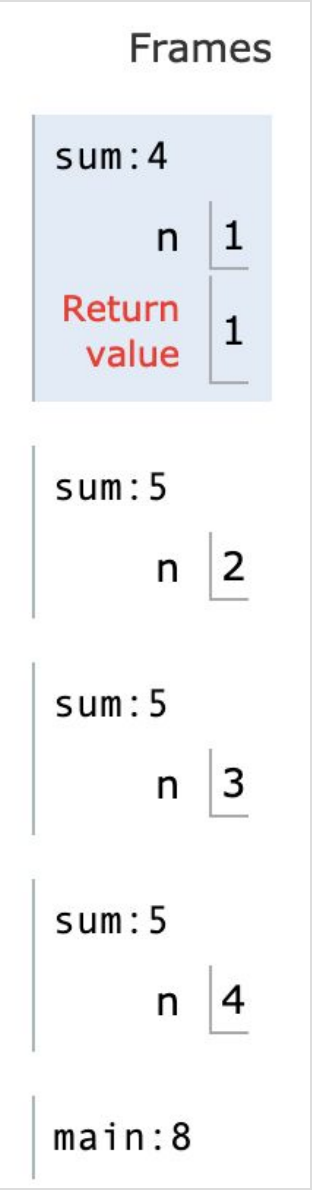
DEMO: VISUALIZATION OF RECURSION

https://cscircles.cemc.uwaterloo.ca/java_visualize/

```
public class Recursion {
   public static int sum(int n) {
      if (n == 1)
         return 1;
      return n + sum(n-1);
    }
    public static void main(String[] args) {
      int result = sum(4);
      System.out.println(result);
    }
}
```

Interactive Java visualizer- shows stack frames during execution.

Copy and paste in this code to see it.



RECAP: RECURSIVE vs ITERATIVE

```
public static int sumRecur(int n){
        if(n == 1)
            return 1;
        else
            return n + sumRecur(n-1);
}
```

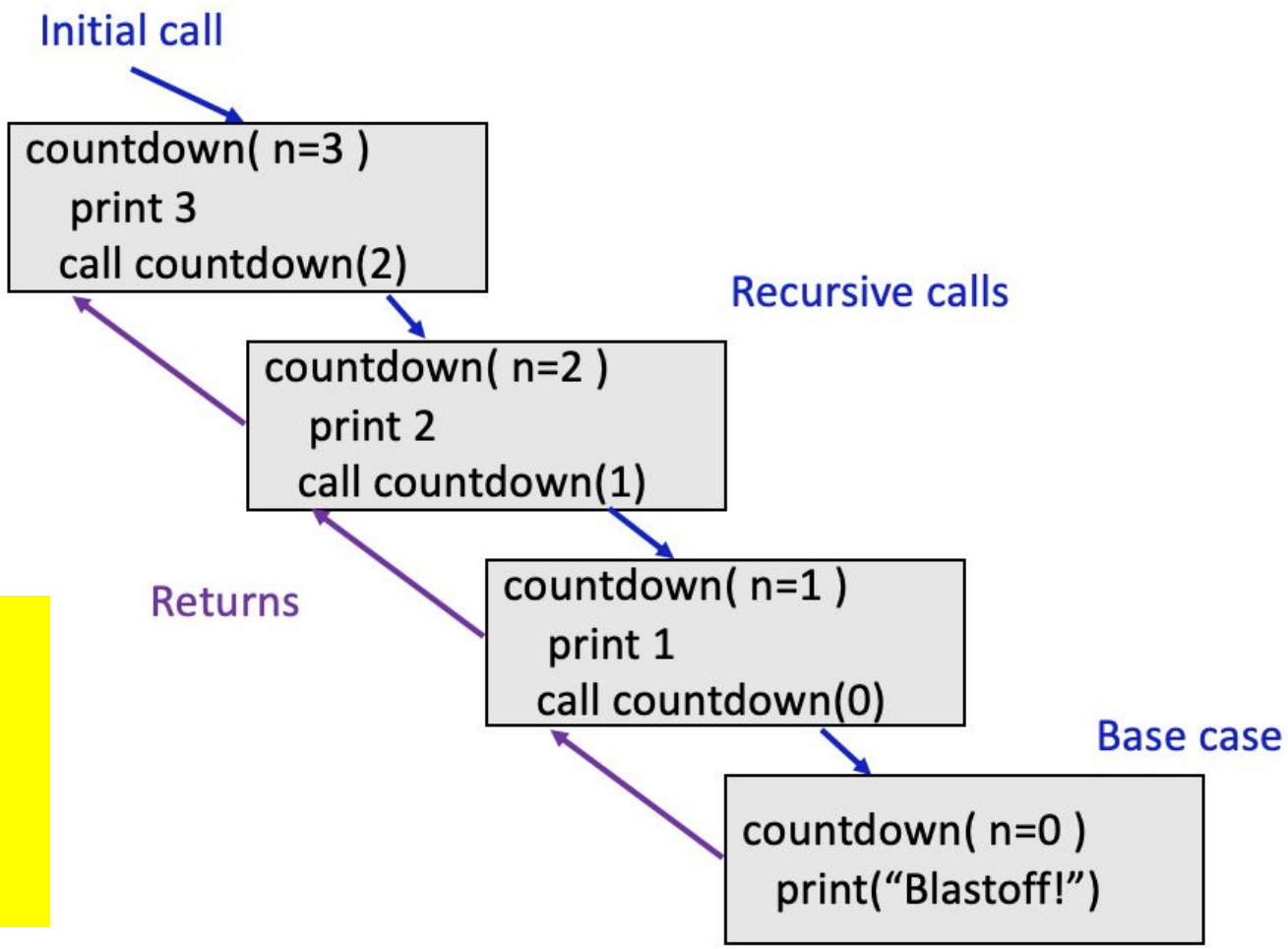
```
public static int sumIter(int n){
        int result = 0;
        for(int i = 1; i <= n; i++)
            result = result + i;
        return result;
}
```

Equivalent **ways of doing this task.**

ANOTHER TRACING EXAMPLE WITH STACK DIAGRAM

```
public static void countdown(int n) {
    if (n == 0) {
        System.out.println("Blastoff!");
    }
    else {
        System.out.println(n);
        countdown(n - 1);
    }
}
```

Stack diagram for countdown(3);



Each method call creates a "stack frame"

Clicker Question 1

What is the output when `n = 3`?

A. `3, 2, 1`
B. `0, 1, 2`
C. `Never terminates`
D. `3, 1, -3`
E. `3, 1, 0`

```
public static void recur (int n){
    if(n==0) {
      return;
   }
    else{
      recur(n - 2);
      System.out.println(n);
    }
}
```

Note return statement in a void method! Different from return at end of method that returns a value of some type

Clicker Question 1 Answer

What is the output when `n = 3`?

A. 3, 2, 1
B. 0, 1, 2
C. **Never terminates**
D. 3, 1, -3
E. 3, 1, 0

```
public static void recur (int n){
    if(n==0) {
       return;
   }
    else{
       recur(n - 2);
       System.out.println(n);
    }
}
```

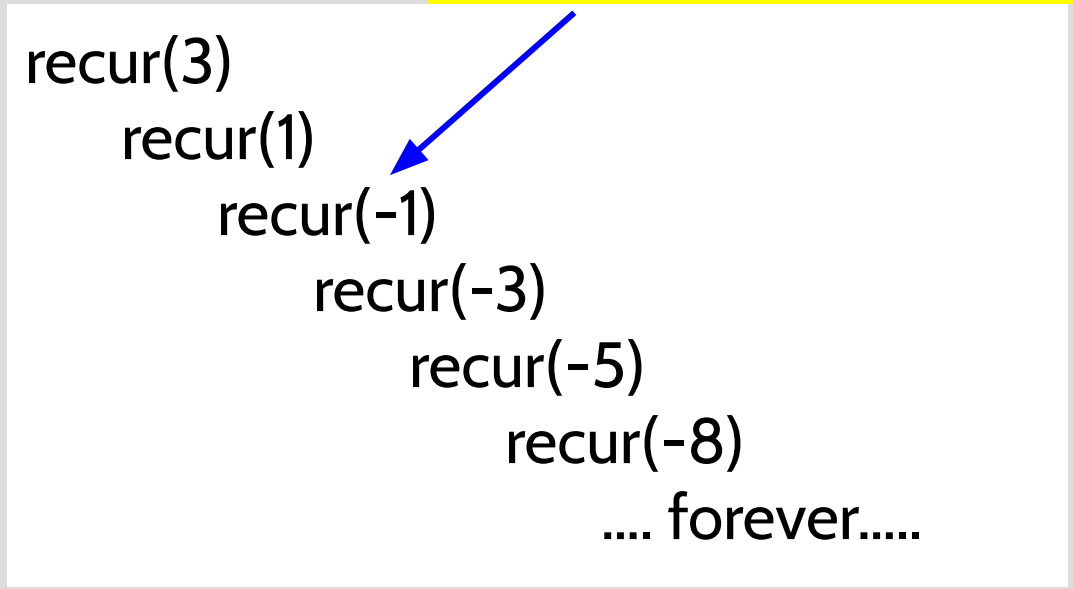
Recursion beginning with an odd or negative argument will "step over" the base case value (0), and will therefore never terminate.

Clicker Question 1 Answer

What is the output when `n = 3`?

```
public static void recur (int n){
    if(n==0) {
      return;
   }
    else{
      recur(n - 2);
      System.out.println(n);
    }
}
```

missed the base case!



RECURSION WITH STRINGS

Write a recursive method to print a String in reverse.
Example: input: "`Hello`" output: "`olleH`"

Strategy:
Iterative: Use a for loop and print characters in reverse order.

Recursive: Identify a base case, then a recursive step that works on a simpler version of the problem.

RECURSION WITH STRINGS

Write a recursive method to print a String in reverse.
Example: input: "`Hello`" output: "`olleH`"

Recursive: Identify a base case, then a recursive step that works on a simpler version of the problem.

Base case: The empty String "" is already "reversed", so just return it. Actually, a String with one character is also "reversed". Let's work with "" so we can accept any String object (not null).

RECURSION WITH STRINGS

Write a recursive method to print a String in reverse.
Example: input: "`Hello`" output: "`olleH`"

Recursive: Identify a base case, then a recursive step that works on a simpler version of the problem.

Base case: The empty String "".
Recursive step: A simpler problem of reversing a String is to reverse a smaller String (one less character).

RECURSION WITH STRINGS

Write a recursive method to print a String in reverse.

Example: input: "Hello" output: "olleH"

Pseudocode:

```
Given String str:
    if length of str = 0  // identifies the empty String
        return;
    else
        reverse (str minus 1st char) // reverse str minus 1st char
        print(1st char of str)
            // note that the print is after the recursive  call
```

RECURSION WITH STRINGS

Write a recursive method to print a String in reverse.

Example: input: “Hello” output: “olleH”

```
reverse(“Hello”)
   reverse(“ello”)
      reverse(“llo”)
         reverse(“lo”)
            reverse(“o”)
               reverse(“”)
            print(‘o’)
         print(‘l’)
      print(‘l’)
   print(‘e’)
print(‘H’)
```

Tracing the execution.

Output that is printed:

“olleH”

Verify algorithm:

- Start with a String of length >=0.
- Stop at length==0.
- Each step is length-1, so *guaranteed* to reach the base case.

What happens if the base case is never reached?

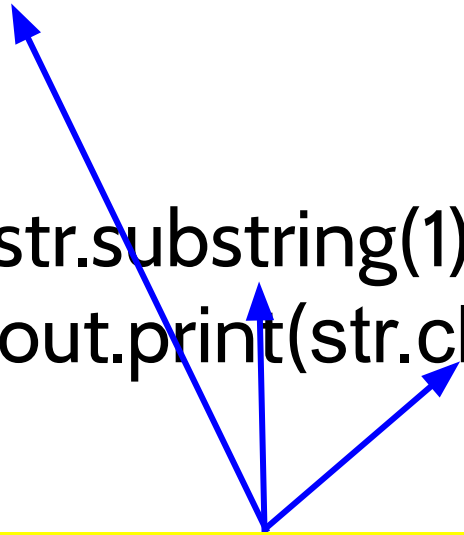
Implementation in Java.

Pseudocode:

```
Given String str:
    if length of str = 0
        return;
    else
        reverse (str minus 1st char)
        print(1st char of str)
```

Do the design work first so you understand how and if the solution works, then worry about Java!

```
public static void reverse(String str) {
    if (str.length() == 0 )
      return;
    else {
      reverse(str.substring(1));
      System.out.print(str.charAt(0))
    }
}
```



Use of String class methods.

Clicker Question 2

What is the output for: `recurString("donkey");`

```
public static void recurString(String s){
      if(s.length() == 0)
         return;
      else {
         System.out.print(s.charAt(0));
         recurString(s.substring(1));
      }
   }
```

A. yeknod
B. Does not stop
C. donke
D. donkey
E. eknod

Clicker Question 2 Answer

What is the output for: `recurString("donkey");`

```
public static void recurString(String s){
      if(s.length() == 0)
         return;
      else {
         System.out.print(s.charAt(0));
         recurString(s.substring(1));
      }
   }
```

A. yeknod
B. Does not stop
C. donke
D. donkey
E. eknod

Trace through this method to see why this is the answer. How does this method differ from the reverse code?

RECURSION - WHY?

Recursion and iteration are equivalent approaches to solving a problem.

Most problems can be solved iteratively, but there are some problems that are easier to solve in a recursive manner.

Sorting Example: a char array {'z', 'd', 'w', 'a', 's'}

base case: one character- it is sorted.

recursive step: divide the array in half, call sort on each half and merge them correctly.

This sorting algorithm called "merge sort" is covered in a data structures course. You are not expected to know it for this course- it is just an example.

RECURSION - KEY SKILLS

Key Skills to know:

- **Be able to write a recursive method like the examples sum and reverse.**

- **Be able to trace a recursive method to understand what it does when executed.**

In order to do these skills you will need to be comfortable with the info presented in these slides and also in the text.

Make sure you go over the simple examples and ask if something isn't clear.

TO-DO LIST

- Complete zyBook chapter 8 exercises.
- For more visualizations on Recursion see this link.