# Recursion I

Methods that call themselves

## Project

- Any questions about the project?
- Recall
  - Minimum Submission due Jan 11 (this Sunday)

## Reminder

- 1st Exam
  - Wednesday
  - Will cover
    - Inheritance
    - Exceptions
  - 10-20 questions
  - Variety of question types
    - Short answer
    - Fill in the code
    - Step through the code
    - Perhaps some multiple choice

## Final Exam – Good news/bad news

- Good news
  - Exam is mid-week and in this building:
    - Wednesday, February 25, 2004
    - 70-3435
- Bad news
  - The time
    - 8:00am – 10:00 am

## Before we begin

- Any questions?

## Recursive Functions

- A recursive function is a function that is defined in terms of itself.
  - Example: factorial

$$n! = \begin{cases} 1 & \text{if } n = 1 \\ n * (n-1)! & \text{otherwise} \end{cases}$$

## Recursive Functions

- Example: Factorial

4! = 4 * 3!

= 4 * (3 * 2!)

= 4 * (3 * (2 * 1!))

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

= 24

## Recursive Methods

- A recursive method is one that can call itself

Non-recursive

```
methodA() {
…
   methodB ();
…
}
```

Recursive

```
methodB() {
…
   methodB ();
…
}
```

## Recursive Methods

- Let's code a recursive function for factorial:

$$n! = \begin{cases} 1 & \text{if } n = 1 \\ n * (n-1)! & \text{otherwise} \end{cases}$$

## Recursive Methods

- Let's code a recursive function for factorial:
- `factorial(N) = 1` if N = 1

`= N * factorial (N-1)`

otherwise

## Recursive Methods

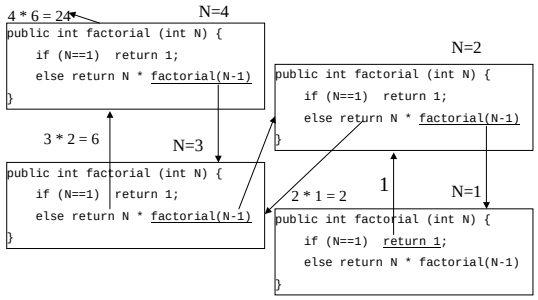
- Let's code a recursive function for factorial:

```
public int factorial (int N)
{
      if (N == 1)  return 1;
      else return N * factorial (N-1);
}
```

## Recursive Methods

- Example: `int fact = factorial (4);`
  - factorial(4) = 4 * factorial(3)
  - = 4 * (3 * factorial(2))
  - = 4 * (3 * (2 * factorial(1)))
  - = 4 * (3 * (2 * (1 * 1)))
  - = 4 * (3 * (2 * 1))
  - = 4 * (3 * 2)
  - = 4 * 6
  - = 24

## Calling recursive functions




## Calling recursive functions

- But can't recursion go on forever?

## Calling recursive functions

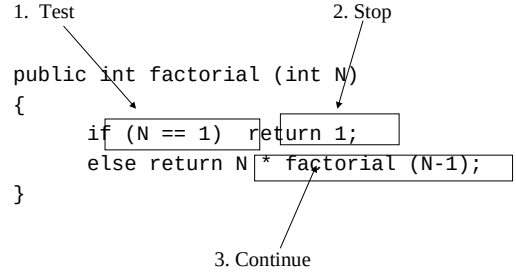
- But can't recursion go on forever?
  - Yes, unless we insure that the recursion will stop for a given condition

```
public int factorial (int N)
{
      if (N == 1)  return 1;
      else return N * factorial (N-1);
}
```

## Components of a recursive methods

- Three necessary components for a recursive method:
  1. A test to stop or continue the recursion
  2. An end case that stops the recursion
  3. A recursive call that continues the recursion.

## Components of a recursive methods




## Iterative Solutions

- Each recursive solution has a corresponding iterative solution that doesn't use recursion.

```
– int factorial2 (int N)
   {
        int fact = 1;
        for (i=1; i <= N; i++)  fact *=
  i;
        return i;
    }
```

## Iterative Solutions

- Then why use recursion?
  - Many times, the iterative solutions are far more complex than the recursive solutions.
  - Many times, it is easy and natural to express a solution using recursion.
  - Questions?

## Another example

- Fibonacci Numbers:

$$fib(n) = \begin{cases} 1 & \text{if } n = 1,2 \\ fib(n-2) + fib(n-1) & \text{otherwise} \end{cases}$$

## Another example

- recursive function for fibonacci:

```
public int fib (int N)
{
      if (N <= 2)  return 1;
      else
        return fib(N-1) + fib (N-2)
}
```

## Another example

- Not a good candidate for recursion

```
public int fib (int N)
{
      if (N <= 2)  return 1;
      else
        return fib(N-1) + fib (N-2)
}
```

Duplicate effort

## When not to use recursion

- The recursive solution results in duplicate computation.
- Questions?

## Recursion using arrays

- findsum – use recursion to find the sum of values in an array x of size n.


Sum = value at n-1 + sum of values from 0 to n-2

## Recursion using arrays

- findsum – use recursion to find the sum of values in an array x of size n.
  - findsum (x, n) = x[n-1] + findsum(x, n-1)

## Calling recursive functions

- But can't recursion go on forever?

## Recursion using arrays

- findsum – use recursion to find the sum of values in an array x of size n.
  - findsum (x, n) = x[n] + findsum(x, n-1)
  - The recursion stops when n = 1
    - The sum of an array of one element is the value of the single element
    - findsum (x, 1) = x[0]

## Recursion using arrays

- findsum – use recursion to find the sum of values in an array x of size n.

```
public int findsum (int[] x, int n)
 {
      if (n == 1) return x[0]
      else
       return x[n-1] + findsum(x, n-1)
 }
```

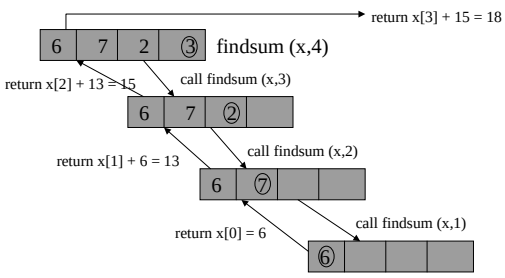
## Recursion using arrays

1. Test
2. Stop

```
public int findsum (int[] x, int n)
 {
      if (n == 1) return x[0]
      else
       return x[n-1] + findsum(x, n-1)
 }
```

3. Continue

## Recursion using arrays



## Questions?

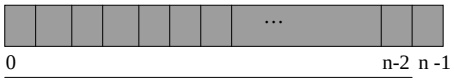
- Let's try one more example

## One last example

- `search()` – given an array of ints, `x`, with length `n`, return a boolean flag indicating if a given integer `i` is in the array

## One last example

- `search()` – return whether a given integer, i, is in an array x of length n




If x[n-1] = = i then return true

otherwise perform a search on the remaining n-1 elements of the array

## One last example

```
public boolean search(int x[], int
  n, int i)
{
    if (x[n-1] == i) return true;
    else
        return search (x, n-1, i);
}
```

## One last example

- But what happens when i is not in x?

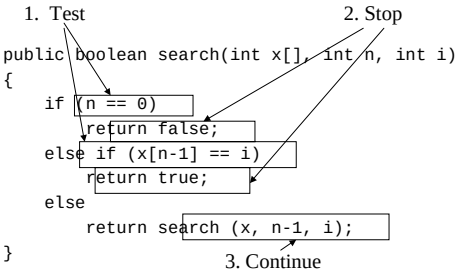
## One last example

- `search()` – return whether a given integer, i, is in an array x of length n
  - The recursion stops when either
    - i is found
    - The length n is equal to 0
      - i will never be found in an array of length 0
      - In this case, false will be returned.

## One last example

```
public boolean search(int x[], int n, int
  i)
{
    if (n == 0)
        return false;
    else if (x[n-1] == i)
        return true;
    else
        return search (x, n-1, i);
}
```

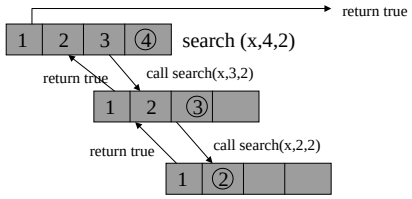
## One last example



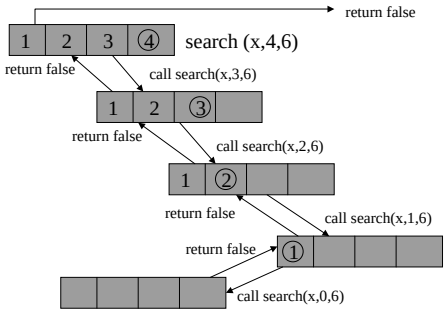

## One last example

- Can you map out the `search` calling sequence for?:
  - x = [1,2,3,4] i=2
  - x = [1,2,3,4] i=5

## Search (x, 4, 2)




## Search (x, 4, 6)




## search

- Questions on search?

## When to use recursion

1. A recursive solution is natural and easy to understand
2. When the recursive solution does not result in excess computation
3. The equivalent iterative solution is more complex.

## Summary

- Recursion – When a method calls itself
- Components of a recursive method
  - Test
  - Stop
  - Continue
- When to use recursion.

## Next time

- The Tower of Hanoi.
- Questions?