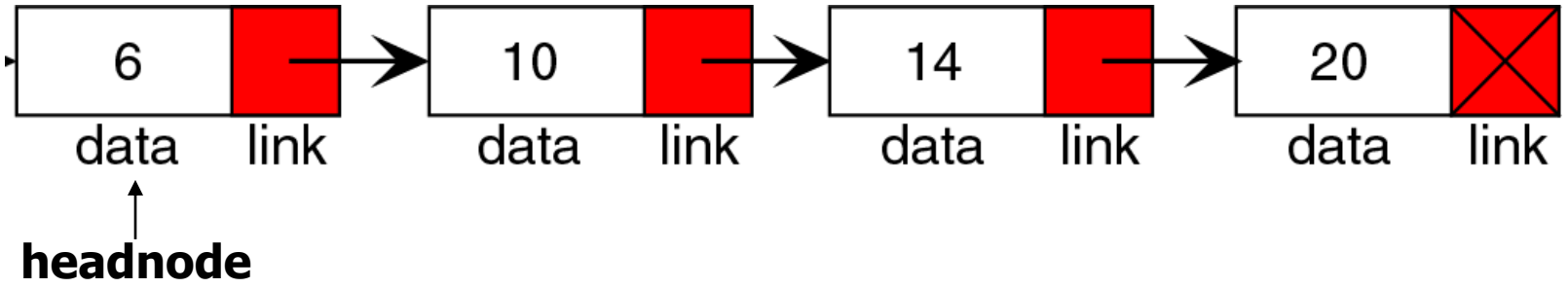
Recursion Application and Examples

Application

- One application of recursion is reversing a list.
- we will implement the same function using recursive techniques.

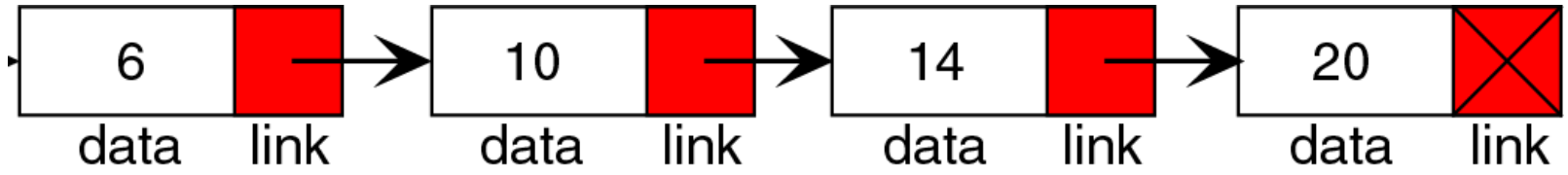
Application

- Let's say we want to reverse the list below.
- The pseudocode to do this is shown on the next slide.



Reverse List Pseudocode

```
printReverse(list) {
    if(null list)
        return;
    else
    printReverse(list->next);
    //Once we're here, the end of the list has
    // been reached
    print(list->data);
    return;
}
```



headnode

printReverse(6)

printReverse(10)

//list->next()

cout<<list->data();

printReverse(14)

6

10

cout<<list->data();

printReverse(20)

14

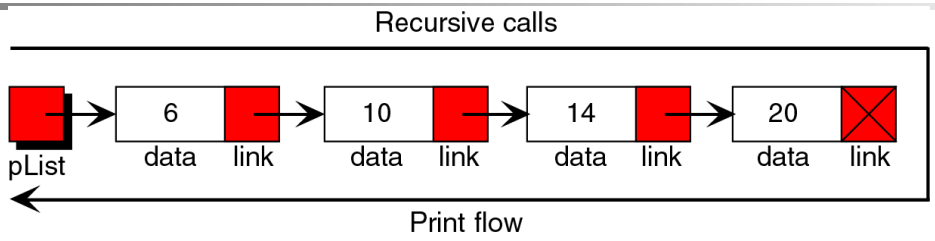
cout<<list->data();

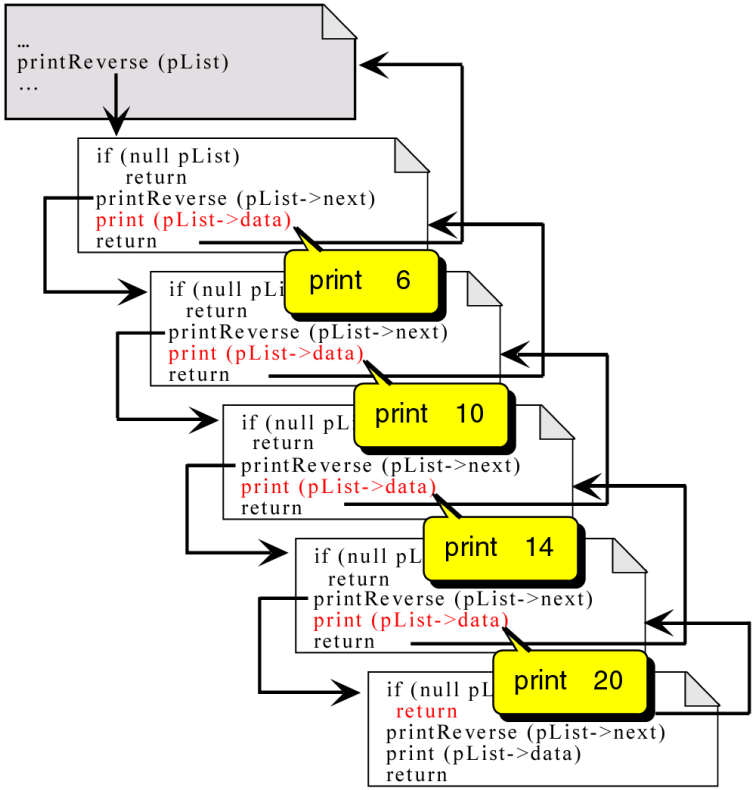
printReverse(null)

20

cout<<list->data();

Algorithm Flow





Example-Fibonanci series

- **Fibonnacci Sequence**
- 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...
- Each element is the sum of the two preceding elements with

  fib(0) = 0

  fib(1) = 1

  fib(n) = n if n ==0 or n ==1

  fib(n) = fib(n-2) + fib(n-1) if n >= 2
- E.g.

Recursive Programming

- **Fibonnacci Sequence (0,1,1,2,3,5,8,13,…..)**

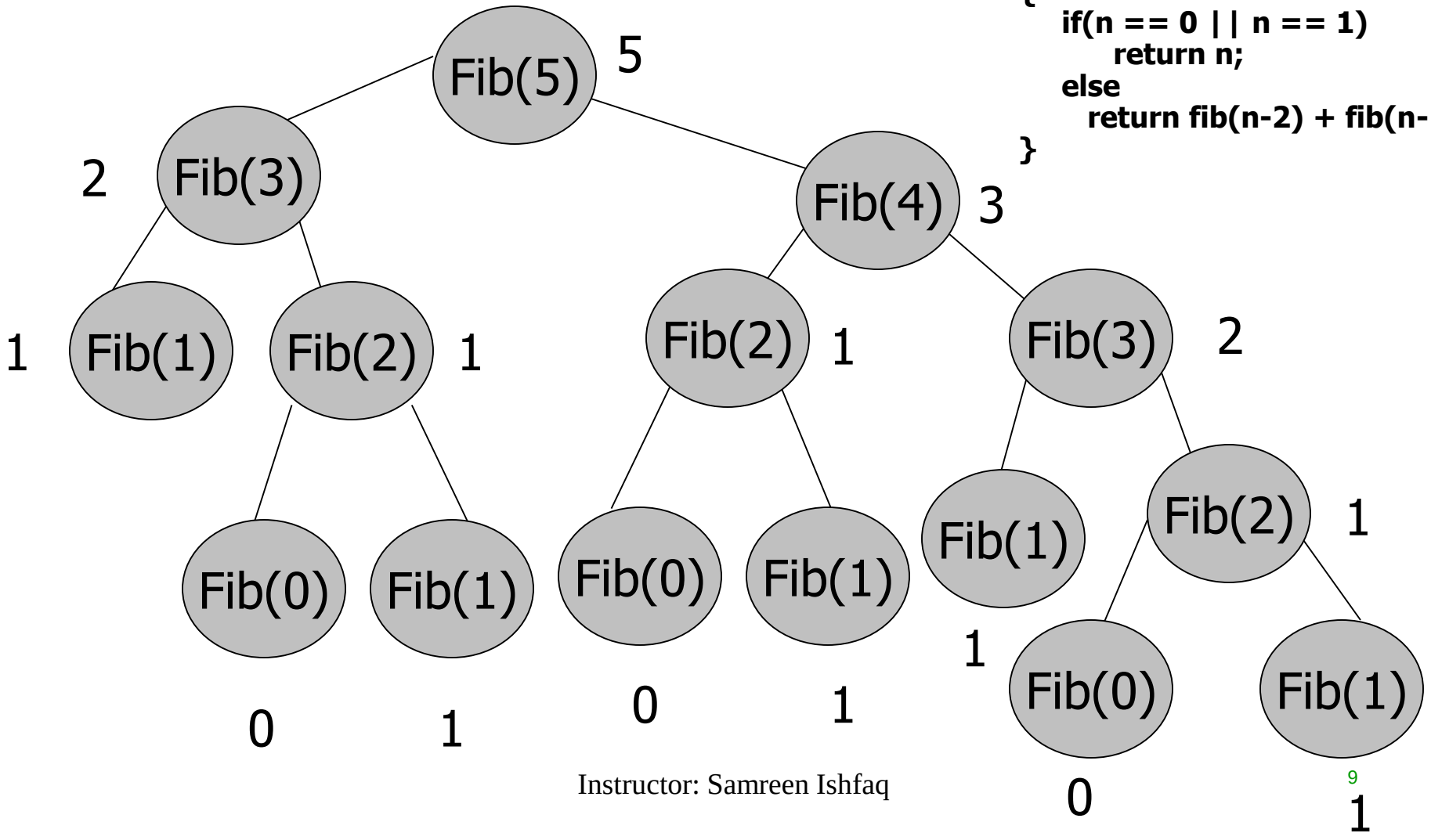
fib(n) = n if n ==0 or n ==1

fib(n) = fib(n-2) + fib(n-1) if n >= 2

```
int fib(int n)
{
    if(n == 0 || n == 1)
        return n;
    else
        return fib(n-2) + fib(n-1)
}
```

Problem To Solved (Fib(5))

```
int fib(int n)
{
    if(n == 0 || n == 1)
        return n;
    else
      return fib(n-2) + fib(n-1);
}
```



Dry Run (Home Task)

- Find fib(7) and how many function calls were executed?

Conclusion

- A recursive solution solves a problem by solving a smaller instance of the same problem.
- It solves this new problem by solving an even smaller instance of the same problem.
- Eventually, the new problem will be so small that its solution will be either obvious or known.
- This solution will lead to the solution of the original problem.

Conclusion

- 4 questions for constructing recursive solutions:
  - How can you define the problem in terms of a smaller problem of the same type?
  - How does each recursive call diminish the size of the problem?
  - What instance of the problem can serve as the base case?
  - As the problem size diminishes, will you reach the base case?

Conclusion

- In general, there is no reason to incur the overhead of recursion when its use does not gain anything.
- Recursion is truly valuable when a problem has no simple iterative solution.

Example – Find

Home Task

- To find an element in an array
- Base case
  - If array is empty, return false
- Recursive step
  - If 1st element of array is given value, return true
  - Skip 1st element and recur on remainder of array

Example – Count

Home Task

- To count # of elements in an array
- Base case
  - If array is empty, return 0
- Recursive step
  - Skip 1st element and recur on remainder of array
  - Add 1 to result