1806ICT Programming Fundamentals

Recursion

1

1

Topics

- Recursion - Definition
- Recursion Examples

2

2

What is Recursion?

- A procedure defined in terms of (simpler versions of) itself
  - Another definition: the problem is expressed in terms of a smaller version of itself
- Components:
  - Base case
  - Recursive definition
  - Convergence to base case

3

3

Example – Counting Down

- Say if we are given a number **n**, we want to print

  n, n-1, n-2, … , 3, 2, 1

- We can do this by using a for loop

```
int n = 5;

for (int i=n; i>0; i--)
{
    printf("%d\n", i);
}
```

4

Example – Counting Down

- A recursive program to do this would be

```
void countDown(int n)
{
	if (n==0)
		return;
	else
	{
		printf("%d\n", n);
		countDown(n-1);
	}
}

int main()
{
	countDown(5);
	return 0;
}
```

5

Recursive Definitions

- Recursive algorithm
  - Algorithm that finds the solution to a given problem by reducing the problem to “smaller” versions of itself
  - Implemented using recursive methods
  - Has one or more base cases
- Base case
  - Does not contain a recursive call
  - The solution is obtained directly
  - Stops the recursion
- Recursive function
  - Function that calls itself
  - Calls a “smaller” version of itself

```
void countDown(int n);

int main()
{
	countDown(5);
	return 0;
}

void countDown(int n)
{
	if (n==0)
		return;
	else
	{
		printf("%d\n", n);
		countDown(n-1);
	}
}
```

6

Recursive Definitions

- General solution
  - Break problem into smaller versions of itself
  - Implement a recursive function in which a "smaller" version of itself is called
  - Must eventually be reduced to a base case that does not contain a recursive call

7

Recursive Definitions

- Directly recursive: a function that calls itself
- Indirectly recursive: a function that calls another function and eventually results in the original function call
- Infinite recursion: the case where every recursive call results in another recursive call
  - Never ending
  - You don't want this!

8

Example: Queue processing

```
procedure ProcessQueue ( queue )
{
   if ( queue not empty ) then
   {
      process first item in queue
      remove first item from queue
      ProcessQueue ( rest of queue )
   }
}
```

9

9

Example: Queue processing

```
procedure ProcessQueue ( queue )
{
   if ( queue not empty ) then
   {
      process first item in queue
      remove first item from queue
      ProcessQueue ( rest of queue )
   }
}
```

Base case: queue is empty

10

Example: Queue processing

```
procedure ProcessQueue ( queue )
{
  if ( queue not empty ) then
  {
    process first item in queue
    remove first item from queue
    ProcessQueue ( rest of queue )
  }
}
```

Recursion: call to ProcessQueue

11

Example: Queue processing

```
procedure ProcessQueue ( queue )
{
  if ( queue not empty ) then
  {
    process first item in queue
    remove first item from queue
    ProcessQueue ( rest of queue )
  }
}
```

Convergence: fewer items in queue

12

Example: Factorial

Given $n \geq 0$:

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

$$0! = 1$$

Example:

$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$

13

13

Example: Factorial

- *Problem:* Write a recursive function **`Factorial(n)`** which computes the value of $n!$
- *Base Case*:

  If $n = 0$ or $n = 1$:

  Factorial(n) = 1

14

14

Example: Factorial

- *Recursion*:

$$n! = n \times \underbrace{(n - 1) \times (n - 2) \times \ldots \times 2 \times 1}_{(n-1)!}$$

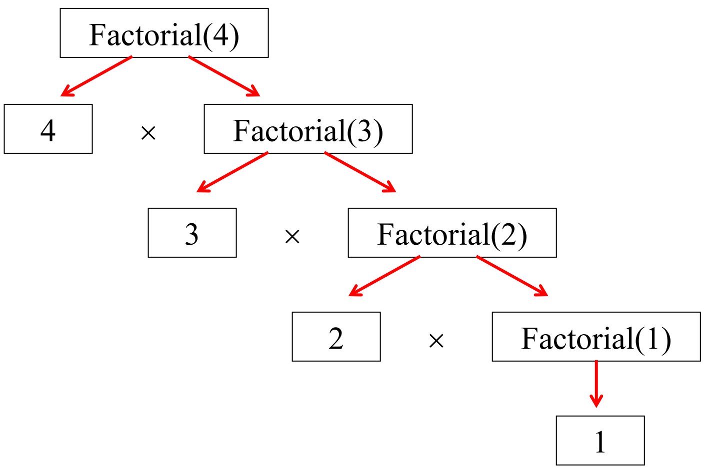
If $n > 1$:
  Factorial(n) = $n \times$ Factorial(n - 1)

15

15

Example: Factorial

- *Convergence*:



16

16

Example: Factorial

The Factorial function can be defined recursively as follows:

Factorial(0) = 1

Factorial(1) = 1

Factorial(n) = $n \times$ Factorial(n - 1)

17

17

Example: Factorial

```
function Factorial ( n )
{
   if ( n is less than or equal to 1 ) then
      return 1
   else
      return n × Factorial ( n – 1 )
}
```

18

18

Example: Factorial

Base case

```
function Factorial ( n )
{
   if ( n is less than or equal to 1 ) then
      return 1
   else
      return n × Factorial ( n – 1 )
}
```

19

19

Example: Factorial

General Case

```
function Factorial ( n )
{
   if ( n is less than or equal to 1 ) then
      return 1
   else
      return n × Factorial ( n – 1 )
}
```

20

20

Example: Factorial

Recursion

```
function Factorial ( n )
{
   if ( n is less than or equal to 1 ) then
      return 1
   else
      return n × Factorial ( n − 1 )
}
```

21

21

Example: Factorial

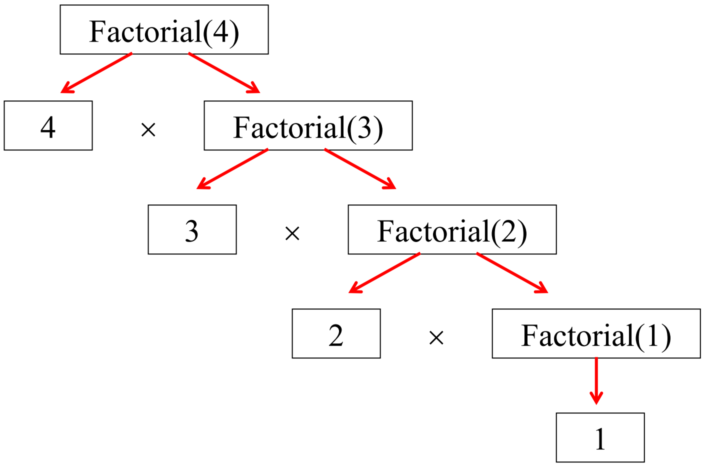
Convergence

```
function Factorial ( n )
{
   if ( n is less than or equal to 1 ) then
      return 1
   else
      return n × Factorial ( n − 1 )
}
```

22

22

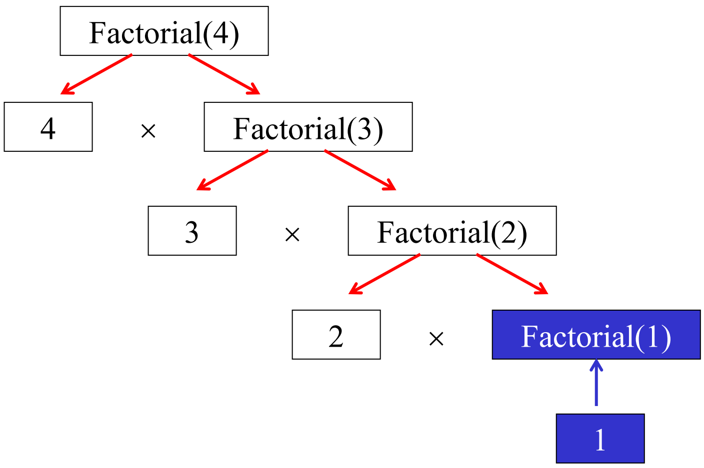
Example: Factorial



23

23

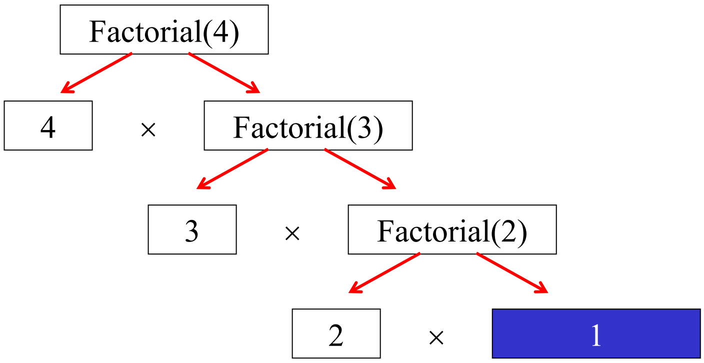
Example: Factorial



24

24

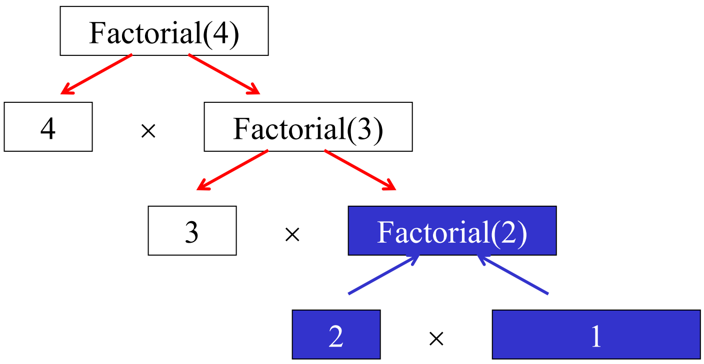
Example: Factorial



25

25

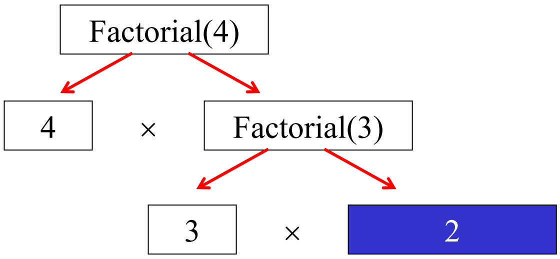
Example: Factorial



26

26

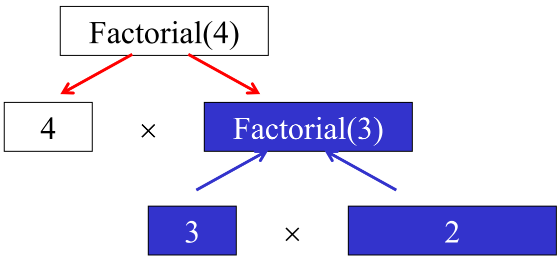
Example: Factorial



27

27

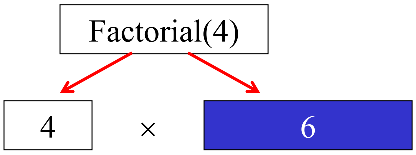
Example: Factorial



28

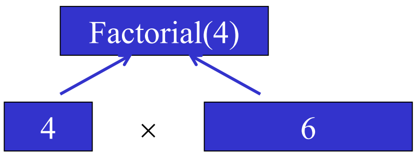
28

Example: Factorial



29

Example: Factorial



30

Example: Factorial

24

31

31

Example:

Computes the factorial of a number
 	
function Factorial (n)
{
if (n is less than or equal to 1)
then
return 1
else
return n × Factorial (n - 1)
}
	`/* Compute the factorial of n */`
	
	`int factorial ( int n )`
	`{`
	`if ( n <= 1 )`
	`{`
	`return 1;`
	`}`
	`else`
	`{`
	`return n * factorial(n-1);`
	`}`
	`}`

32

32

Example: "Frames" during calculation of **`factorial(4)`**

```
printf("%d", factorial(4));
```

33

33

Example: "Frames" during calculation of **`factorial(4)`**

```
printf("%d", factorial(4));
```

```
int factorial ( int n )            n: 4
{
   if ( 4 <= 1 )
   {
      return 1;
   }
   else
   {
      return 4 * factorial( 4 - 1 );
   }
}
```

34

34

Example: "Frames" during calculation of **`factorial(4)`**

```
printf("%d", factorial(4));
```

```
int factorial ( int n )          n: 4
{
    if ( 4 <= 1 )
    {
        return 1;
    }
    else
    {
        return 4 * factorial( 3 );
    }
}
```

35

35

Example: "Frames" during calculation of **`factorial(4)`**

```
printf("%d", factorial(4));
```

```
int factorial ( int n )          n: 3
{
    if ( 3 <= 1 )
    {
        return 1;
    }
    else
    {
        return 3 * factorial( 3 - 1 );
    }
}
```

36

36

Example: "Frames" during calculation of `factorial(4)`

```
printf("%d", factorial(4));
```

```
int factorial ( int n )                n: 3
{
    if ( 3 <= 1 )
    {
        return 1;
    }
    else
    {
        return 3 * factorial( 2 );
    }
}
```

37

37

Example: "Frames" during calculation of `factorial(4)`

```
printf("%d", factorial(4));
```

```
int factorial ( int n )                n: 2
{
    if ( 2 <= 1 )
    {
        return 1;
    }
    else
    {
        return 2 * factorial( 2 - 1 );
    }
}
```

38

38

Example: "Frames" during calculation of `factorial(4)`

```
printf("%d", factorial(4));
```

```
int factorial ( int n )        n: 2
{
    if ( 2 <= 1 )
    {
        return 1;
    }
    else
    {
        return 2 * factorial( 1 );
    }
}
```

39

39

Example: "Frames" during calculation of `factorial(4)`

```
printf("%d", factorial(4));
```

```
int factorial ( int n )        n: 1
{
    if ( 1 <= 1 )
    {
        return 1;
    }
    else
    {
        return n * factorial( n - 1 );
    }
}
```

40

40

Example: "Frames" during calculation of **`factorial(4)`**

```
printf("%d", factorial(4));
```

```
int factorial ( int n )                n: 1
{
   if ( 1 <= 1 )
   {
      return 1 ;
   }
   else
   {
      return n * factorial( n - 1 );
   }
}
```

Base case:

`factorial(1) is 1`

41

41

Example: "Frames" during calculation of **`factorial(4)`**

```
printf("%d", factorial(4));
```

```
int factorial ( int n )                n: 2
{
   if ( 2 <= 1 )
   {
      return 1;
   }
   else
   {
      return 2 * factorial( 1 );
   }
}
```

1

42

42

Example: "Frames" during calculation of **`factorial(4)`**

```
printf("%d", factorial(4));
```

```
int factorial ( int n )          n: 2
{
    if ( 2 <= 1 )
    {
        return 1;
    }
    else
    {
        return 2 * factorial( 2 - 1 );
    }
}
```

1

43

43

Example: "Frames" during calculation of **`factorial(4)`**

```
printf("%d", factorial(4));
```

```
int factorial ( int n )          n: 2
{
    if ( 2 <= 1 )
    {
        return 1;
    }
    else
    {
        return 2 ;
    }
}
```

factorial(2) is 2

44

44

Example: "Frames" during calculation of **`factorial(4)`**

```
printf("%d", factorial(4));
```

```
int factorial ( int n )          n: 3
{
    if ( 3 <= 1 )
    {
        return 1;
    }
    else
    {
        return 3 * factorial( 2 );
    }
}
```

$\underbrace{\texttt{factorial(2)}}_{2}$

45

45

Example: "Frames" during calculation of **`factorial(4)`**

```
printf("%d", factorial(4));
```

```
int factorial ( int n )          n: 3
{
    if ( 3 <= 1 )
    {
        return 1;
    }
    else
    {
        return 3 * factorial( 3 - 1 );
    }
}
```

$\underbrace{\texttt{factorial(3 - 1)}}_{2}$

46

46

Example: "Frames" during calculation of **`factorial(4)`**

```
printf("%d", factorial(4));
```

```
int factorial ( int n )        n: 3
{
    if ( 3 <= 1 )
    {
        return 1;                  factorial(3) is 6
    }
    else
    {
        return          6          ;
    }
}
```

47

47

Example: "Frames" during calculation of **`factorial(4)`**

```
printf("%d", factorial(4));
```

```
int factorial ( int n )        n: 4
{
    if ( 4 <= 1 )
    {
        return 1;
    }
    else
    {
        return 4 * factorial( 3 );
    }                  6
}
```

48

48

Example: "Frames" during calculation of `factorial(4)`

```
printf("%d", factorial(4));
```

```
int factorial ( int n )                n: 4
{
    if ( 4 <= 1 )
    {
        return 1;
    }
    else
    {
        return 4 * factorial( 4 - 1 );
    }
}
```

6

49

49

Example: "Frames" during calculation of `factorial(4)`

```
printf("%d", factorial(4));
```

```
int factorial ( int n )                n: 4
{
    if ( 4 <= 1 )
    {
        return 1;
    }
    else
    {
        return 24 ;
    }
}
```

factorial(4) is 24

50

50

Example: "Frames" during calculation of **`factorial(4)`**

```
printf("%d", factorial(4));
```

$\underbrace{\texttt{factorial(4)}}_{24}$

Output: 24

51

51

Example: testprog.c

```
#include <stdio.h>

/* Main program for testing factorial() function */

int main(void)
{
    int n;

    printf("Please enter n: ");
    scanf("%d", &n);

    printf("%d! is %d\n", n, factorial(n));

    return 0;
}
```

52

52

Example: Fibonacci

- A series of numbers which
  - begins with 0 and 1
  - every subsequent number is the sum of the previous two numbers
- 0, 1, 1, 2, 3, 5, 8, 13, 21,...
- Write a recursive function which computes the n-th number in the series (n = 0, 1, 2,...)

53

53

Example: Fibonacci

The Fibonacci series can be defined recursively as follows:

Fibonacci(0) = 0

Fibonacci(1) = 1

Fibonacci(n) = Fibonacci(n - 2) + Fibonacci(n - 1)

54

54

Example: fibonacc.c

```
function Fibonacci ( n )
{
  if ( n is less than or equal to 1 ) then
    return n
  else
    return Fibonacci ( n - 2 ) + Fibonacci ( n - 1 )
}
```

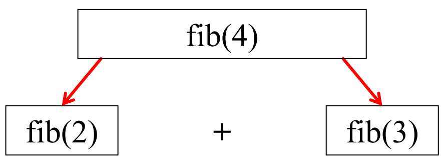
```
/* Compute the n-th Fibonacci number,
   when=0,1,2,... */

long fib ( long n )
{
   if ( n <= 1 )
      return n ;
   else
      return fib( n - 2 ) + fib( n - 1 );
}
```

55

55

Example: Computation of `fib(4)`

```
long fib ( long 4 )
{
   if ( 4 <= 1 )
      return n ;
   else
      return fib( 4 - 2 ) + fib( 4 - 1 );
}
```

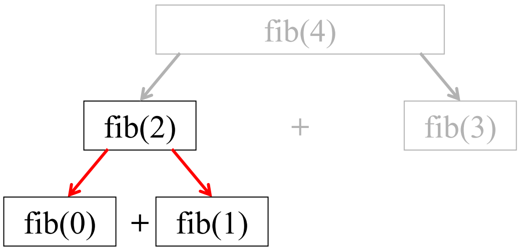


56

56

Example: Computation of `fib(4)`

```
long fib ( long 2 )
{
    if ( 2 <= 1 )
        return n ;
    else
        return fib( 2 - 2 ) + fib( 2 - 1 );
}
```

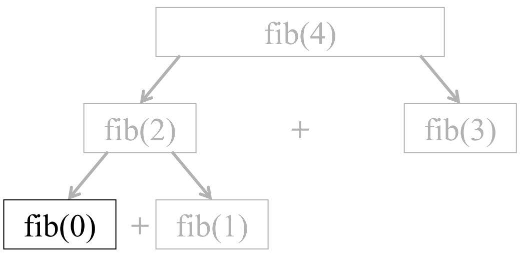


57

57

Example: Computation of `fib(4)`

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

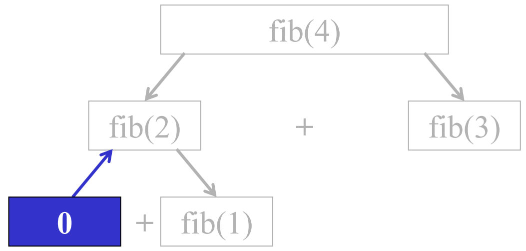


58

58

Example: Computation of `fib(4)`

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

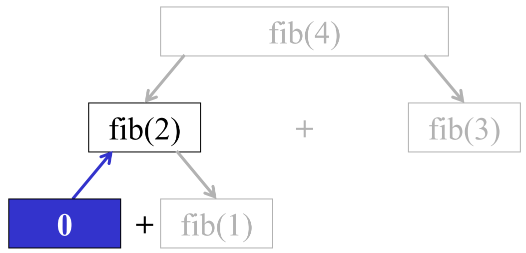


59

59

Example: Computation of `fib(4)`

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

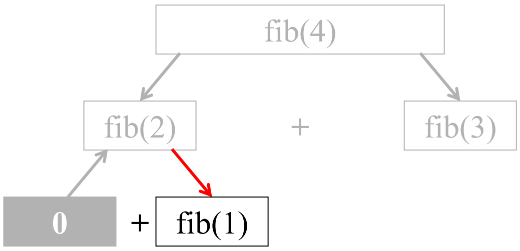


60

60

Example: Computation of `fib(4)`

```
long fib ( long 1 )
{
    if ( 1 <= 1 )
        return 1 ;
    else
        return fib( n - 2 ) + fib( n - 1 );
}
```

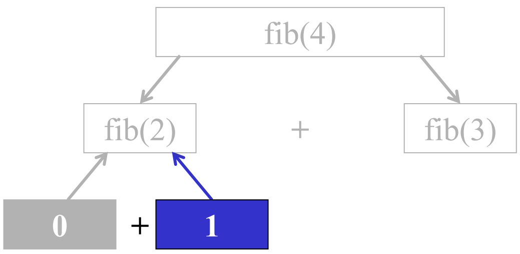


61

61

Example: Computation of `fib(4)`

```
long fib ( long 1 )
{
    if ( 1 <= 1 )
        return 1 ;
    else
        return fib( n - 2 ) + fib( n - 1 );
}
```

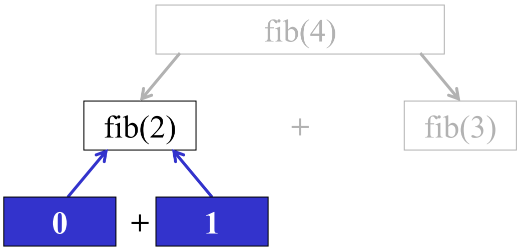


62

62

Example: Computation of `fib(4)`

```
long fib ( long 2 )
{
    if ( 2 <= 1 )
        return n ;
    else
        return      0      +      1      ;
}
```

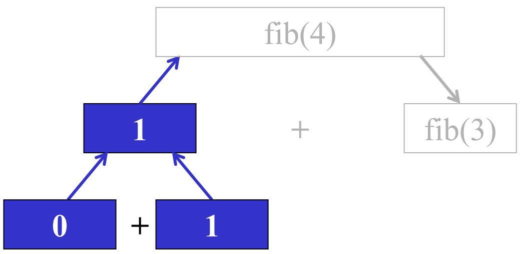


63

63

Example: Computation of `fib(4)`

```
long fib ( long 2 )
{
    if ( 2 <= 1 )
        return n ;
    else
        return      0      +      1      ;
}
```

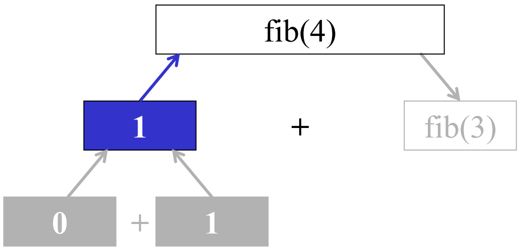


64

64

Example: Computation of `fib(4)`

```
long fib ( long 4 )
{
    if ( 4 <= 1 )
        return n ;
    else
        return      1      + fib( 4 - 1 );
}
```

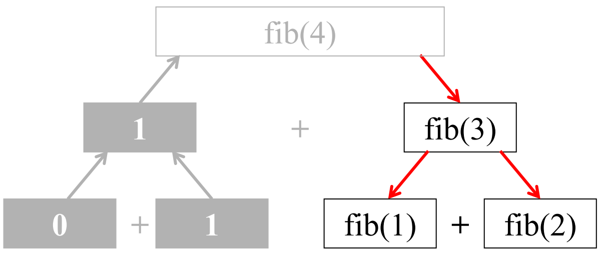


65

65

Example: Computation of `fib(4)`

```
long fib ( long 3 )
{
    if ( 3 <= 1 )
        return n ;
    else
        return fib( 3 - 2 ) + fib( 3 - 1 );
}
```

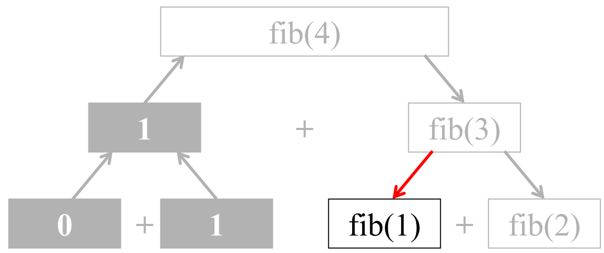


66

66

Example: Computation of `fib(4)`

```
long fib ( long 1 )
{
   if ( 1 <= 1 )
      return 1 ;
   else
      return fib( n - 2 ) + fib( n - 1 );
}
```

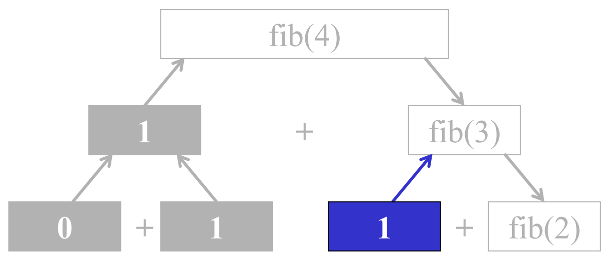


67

67

Example: Computation of `fib(4)`

```
long fib ( long 1 )
{
   if ( 1 <= 1 )
      return 1 ;
   else
      return fib( n - 2 ) + fib( n - 1 );
}
```

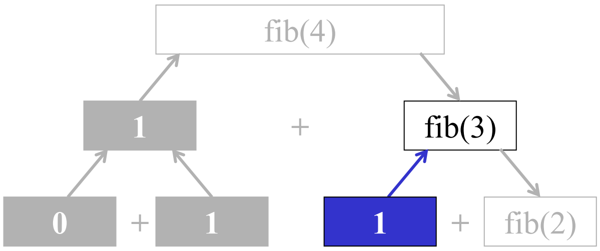


68

68

Example: Computation of **fib(4)**

```
long fib ( long 3 )
{
   if ( 3 <= 1 )
      return n ;
   else
      return        1        + fib( 3 - 1 );
}
```

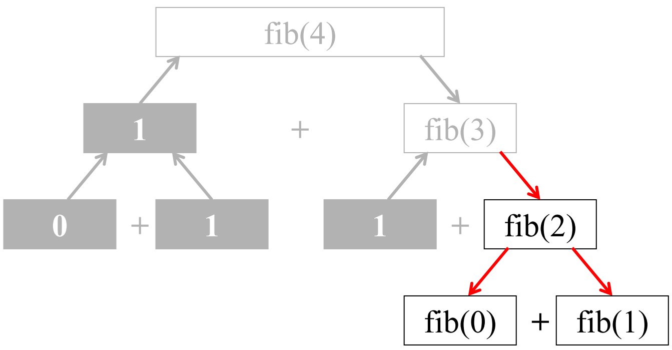


69

69

Example: Computation of **fib(4)**

```
long fib ( long 2 )
{
   if ( 2 <= 1 )
      return n ;
   else
      return fib( 2 - 2 ) + fib( 2 - 1 );
}
```

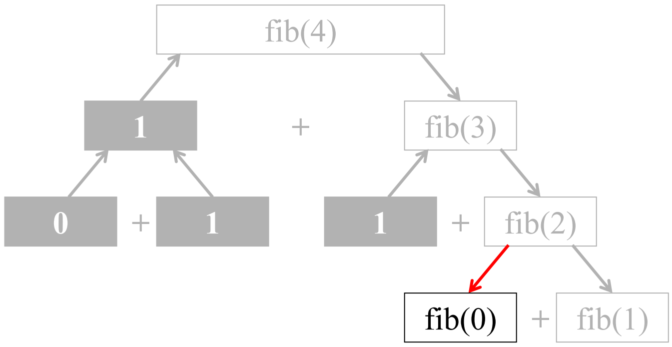


70

70

Example: Computation of `fib(4)`

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

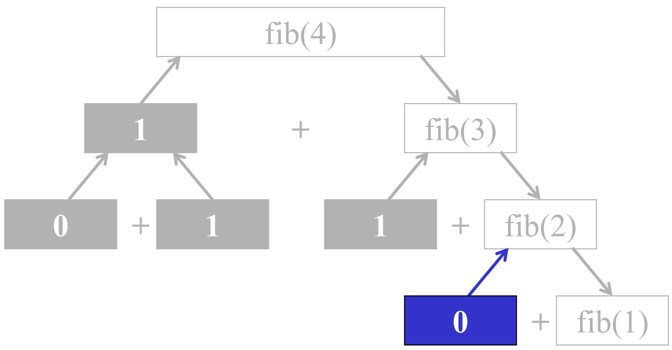


71

71

Example: Computation of `fib(4)`

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

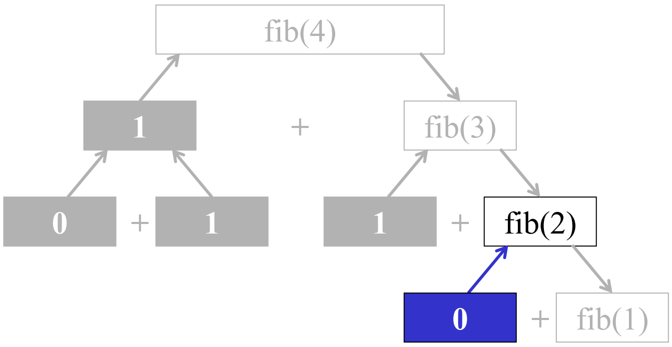


72

72

Example: Computation of `fib(4)`

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

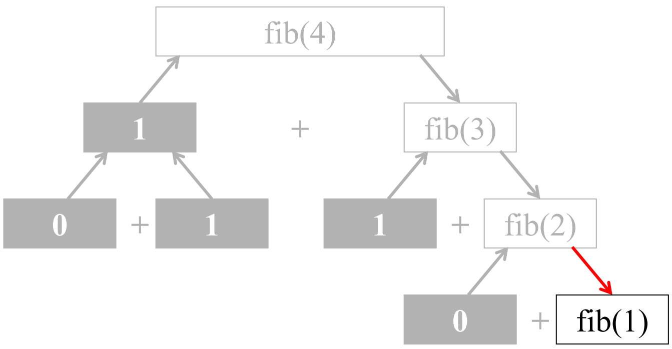


73

73

Example: Computation of `fib(4)`

```
long fib ( long 1 )
{
   if ( 1 <= 1 )
      return 1 ;
   else
      return fib( n - 2 ) + fib( n - 1 );
}
```

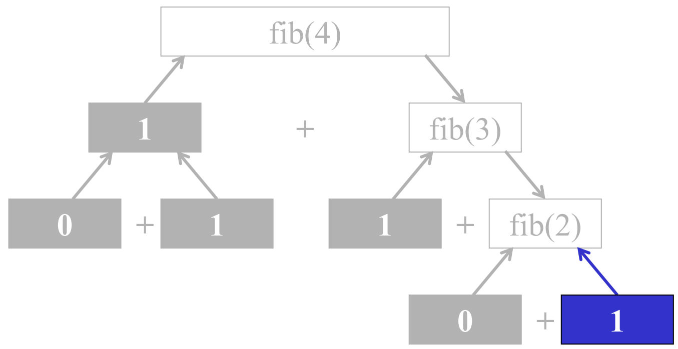


74

74

Example: Computation of `fib(4)`

```
long fib ( long 1 )
{
    if ( 1 <= 1 )
        return 1 ;
    else
        return fib( n - 2 ) + fib( n - 1 );
}
```

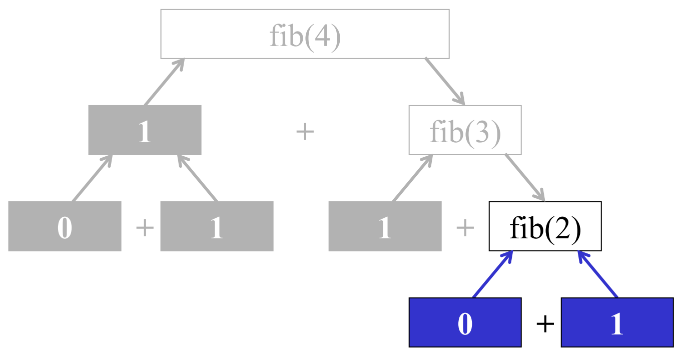


75

75

Example: Computation of `fib(4)`

```
long fib ( long 2 )
{
    if ( 2 <= 1 )
        return n ;
    else
        return          0          +          1          ;
}
```

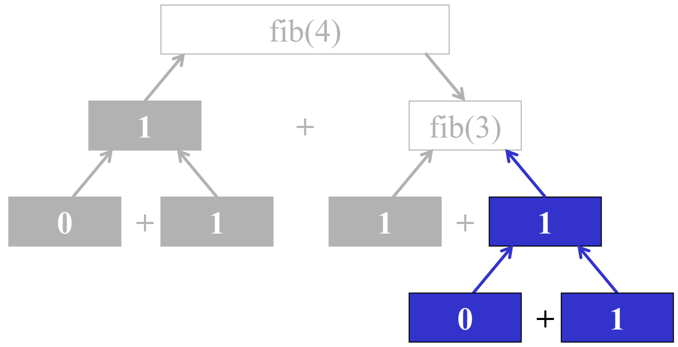


76

76

Example: Computation of `fib(4)`

```
long fib ( long 2 )
{
    if ( 2 <= 1 )
        return n ;
    else
        return    0    +    1    ;
}
```

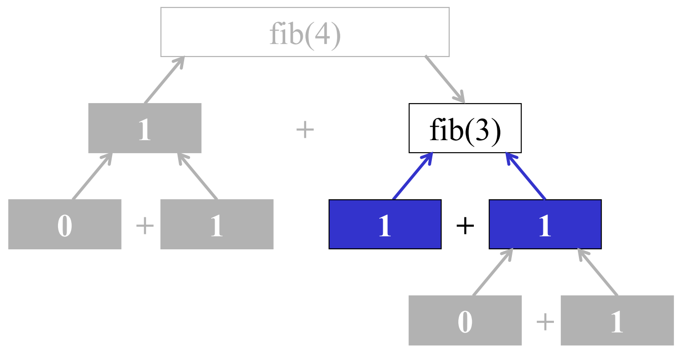


77

77

Example: Computation of `fib(4)`

```
long fib ( long 3 )
{
    if ( 3 <= 1 )
        return n ;
    else
        return    1    +    1    ;
}
```

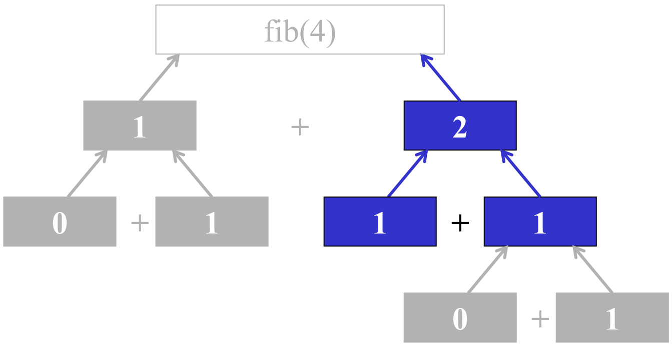


78

78

Example: Computation of `fib(4)`

```
long fib ( long 3 )
{
    if ( 3 <= 1 )
        return n ;
    else
        return    1    +    1    ;
}
```

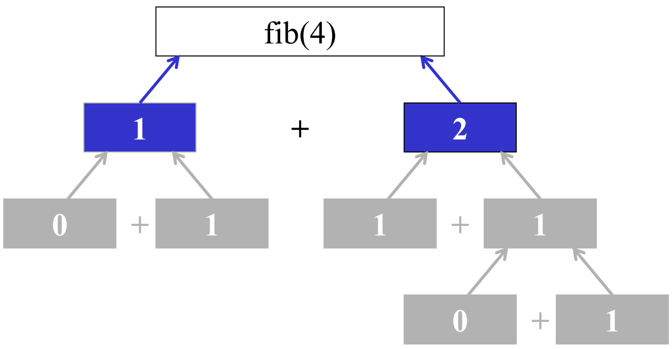


79

79

Example: Computation of `fib(4)`

```
long fib ( long 4 )
{
    if ( 4 <= 1 )
        return n ;
    else
        return    1    +    2    ;
}
```

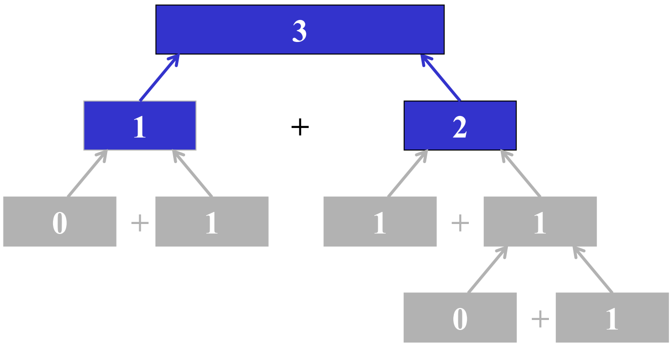


80

80

Example: Computation of `fib(4)`

```
long fib ( long 4 )
{
   if ( 4 <= 1 )
      return n ;
   else
      return      1      +      2      ;
}
```

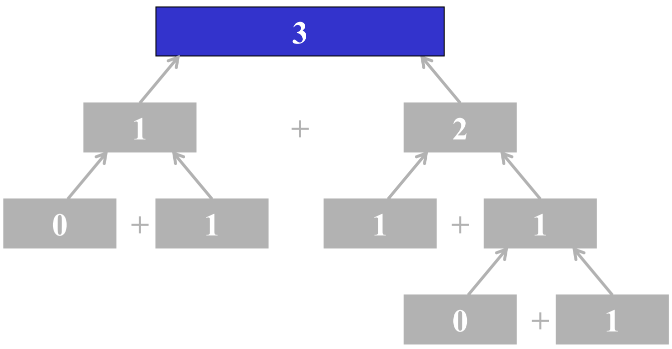


81

81

Example: Computation of `fib(4)`

Thus, **`fib(4)`** returns the value 3.



82

82

Example: fibonacc.c

Sample **`main()`** for testing the **`fib()`** function:

```
int main(void)
{
   long number;

   printf("Enter number: ");
   scanf("%ld", &number);

   printf("Fibonacci(%ld) = %ld\n",
                     number, fib(number));

   return 0;
}
```

83

83

Example: Towers of Hanoi

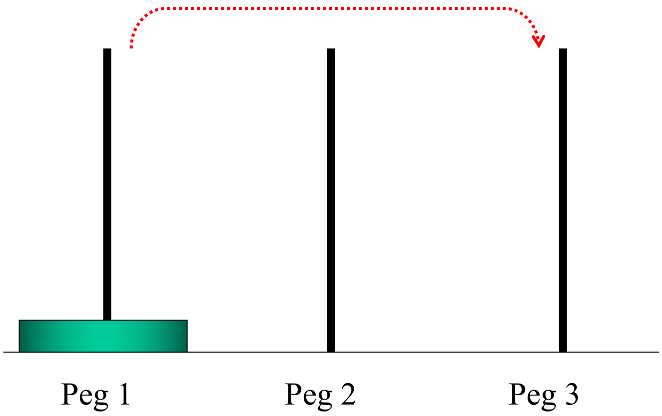
- A classic problem
- Three pegs
- *N* discs, arranged bottom to top by decreasing size
- Objective: Move the discs from peg 1 to peg 3
- Two constraints:
  - One disk is moved at a time
  - No larger disc can be placed above a smaller disk
- Write a program which will print the precise sequence of peg-to-peg disc transfers.

84

84

Example: Towers of Hanoi

Base case: $N = 1$ Peg 1 → Peg 3

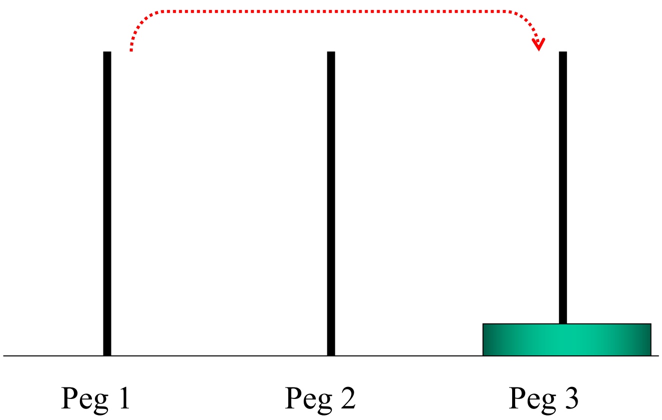


85

85

Example: Towers of Hanoi

Base case: $N = 1$ Peg 1 → Peg 3



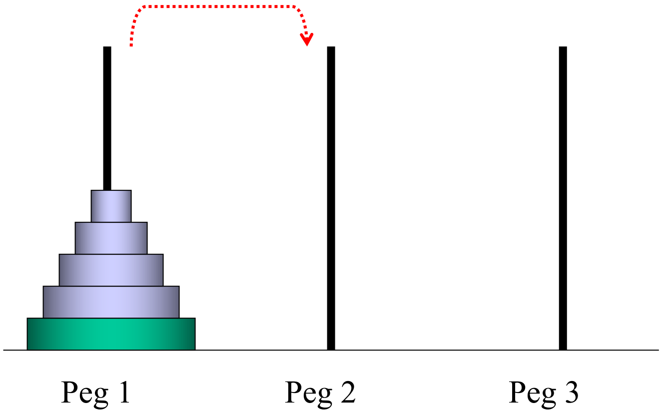
86

86

Example: Towers of Hanoi

Recursion: $N > 1$

Top N-1 discs: Peg 1 $\rightarrow$ Peg 2



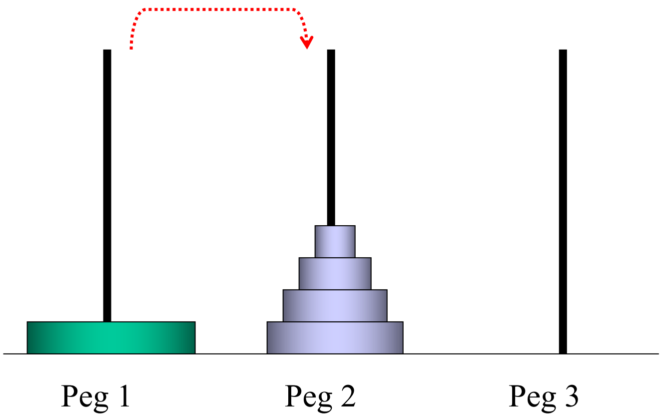
87

87

Example: Towers of Hanoi

Recursion: $N > 1$

Top N-1 discs: Peg 1 $\rightarrow$ Peg 2

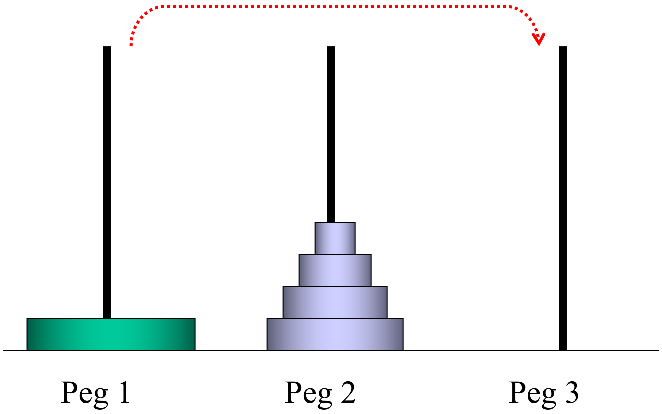


88

88

Example: Towers of Hanoi

Recursion: $N > 1$ Largest disc: Peg 1 → Peg 3

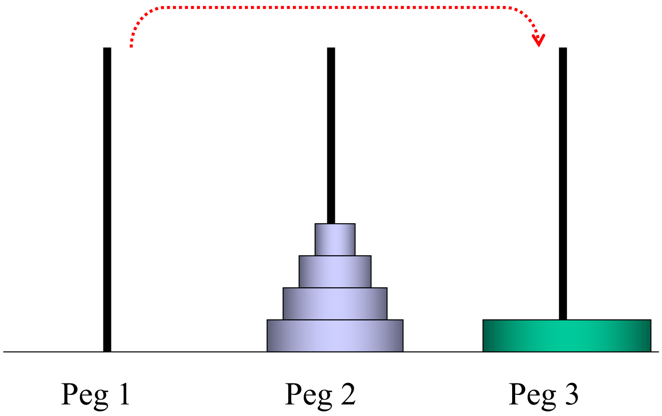


89

89

Example: Towers of Hanoi

Recursion: $N > 1$ Largest disc: Peg 1 → Peg 3



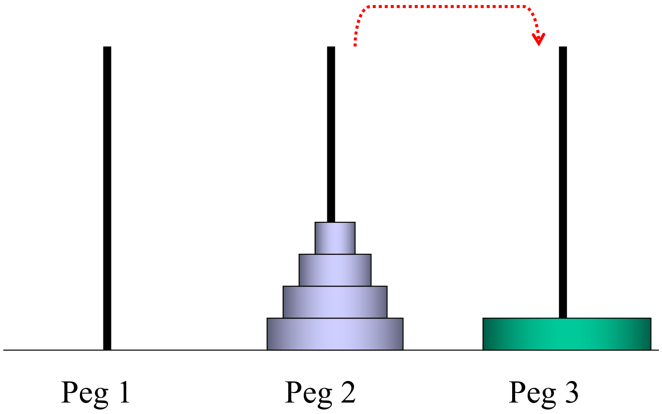
90

90

Example: Towers of Hanoi

Recursion: $N > 1$

Top N-1 discs: Peg 2 $\rightarrow$ Peg 3



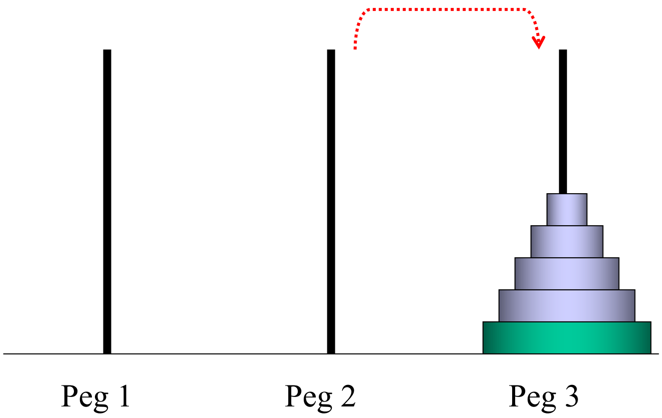
91

91

Example: Towers of Hanoi

Recursion: $N > 1$

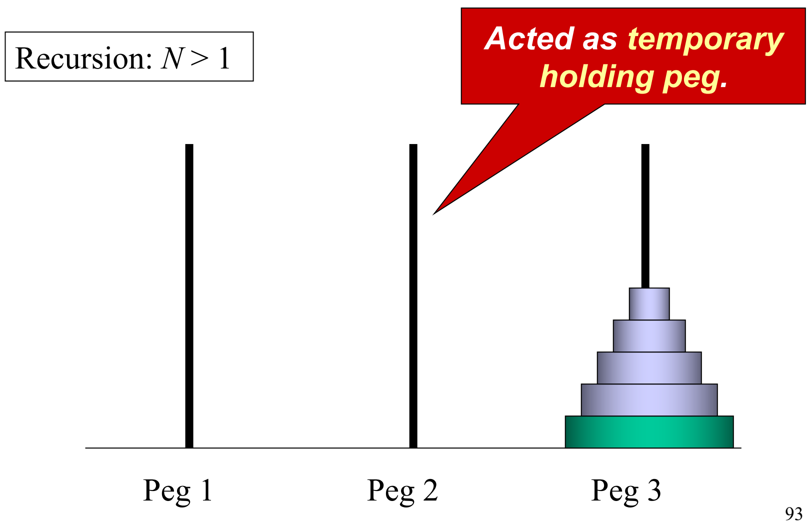
Top N-1 discs: Peg 2 $\rightarrow$ Peg 3



92

92

Example: Towers of Hanoi



93

93

Example: Towers of Hanoi

- Q: But how do you move those N-1 discs from Peg 1 to the temporary holding peg?
- A: Recursively, of course!
  - E.g., "move N-1 discs from Peg 1 to Peg 2 using Peg 3 as temporarily holding area," and so on.
- Denote:
  - *fromPeg*, *toPeg*
  - Temporary holding peg: *otherPeg*

94

94

Example: Towers of Hanoi

```
procedure ToH ( numDiscs, fromPeg, toPeg )
{
  if ( numDiscs is 1 ) then
    output fromPeg, "->", toPeg
  else
  {
    otherPeg = /* determine otherPeg */
    ToH ( numDiscs - 1, fromPeg, otherPeg )
    output fromPeg, "->", toPeg
    ToH ( numDiscs - 1, otherPeg, toPeg )
  }
}
```

Base case

95

95

Example: Towers of Hanoi

```
procedure ToH ( numDiscs, fromPeg, toPeg )
{
  if ( numDiscs is 1 ) then
    output fromPeg, "->", toPeg
  else
  {
    otherPeg = /* determine temporary holding peg here */
    ToH ( numDiscs - 1, fromPeg, otherPeg )
    output fromPeg, "->", toPeg
    ToH ( numDiscs - 1, otherPeg, toPeg )
  }
}
```

Recursion

96

Example: Towers of Hanoi

```
procedure ToH ( numDiscs, fromPeg, toPeg )
{
  if ( numDiscs is 1 ) then
    output fromPeg, "->", toPeg
  else
  {
    otherPeg = theOtherPeg(fromPeg, toPeg)
    ToH(numDiscs - 1, fromPeg, otherPeg)
    output fromPeg, "->", toPeg
    ToH(numDiscs - 1, otherPeg, toPeg)
  }
}
```

97

97

Example: Towers of Hanoi

```
function theOtherPeg ( pegA, pegB )
{
  if ( pegA is 1 ) then
  {
    if ( pegB is 2 ) then
      return 3
    else
      return 2
  }
  else if ( pegA is 2 ) then
  {
    if ( pegB is 1 ) then
      return 3
    else
      return 1;
  }
```

```
  else if ( pegA is 3 ) then
  {
    if ( pegB is 2 ) then
      return 1
    else
      return 2;
  }
```

Solution 1

98

Example: Towers of Hanoi

fromPeg	*toPeg*	*otherPeg*
1	2	3
1	3	2
2	1	3
2	3	1
3	2	1
3	1	2

otherPeg is always 6 - (*fromPeg* + *toPeg*)

99

99

Example: Towers of Hanoi

```
function theOtherPeg (fromPeg, toPeg )
{
  return ( 6 - fromPeg - toPeg )
}
```

Solution 2

100

100

Example: Towers of Hanoi

```
procedure ToH ( numDiscs, fromPeg, toPeg )
{
  if ( numDiscs is 1 ) then
    output fromPeg, "->", toPeg
  else
  {
    otherPeg = theOtherPeg(fromPeg, toPeg)
    ToH(numDiscs - 1, fromPeg, otherPeg)
    output fromPeg, "->", toPeg
    ToH(numDiscs - 1, otherPeg, toPeg)
  }
}
```

101

101

Example: Towers of Hanoi

```
procedure ToH ( numDiscs, fromPeg, toPeg )
{
  if ( numDiscs is 1 ) then
    output fromPeg, "->", toPeg
  else
  {
    otherPeg = theOtherPeg(fromPeg, toPeg)
    ToH(numDiscs - 1, fromPeg, otherPeg)
    output fromPeg, "->", toPeg
    ToH(numDiscs - 1, otherPeg, toPeg)
  }
}
```

Convergence

102

102

```c
/* Given two pegs, this function determines the other peg. */

int theOtherPeg ( int fromPeg, int toPeg )
{
   return (6 - fromPeg - toPeg);
}

/* This functions prints out the precise sequence of peg-to-peg disc
 * transfers to solve the Towers of Hanoi problem. */

void ToH ( int numDiscs, int fromPeg, int toPeg )
{
   int otherPeg;

   if ( numDiscs == 1 )
   {
      printf("%d -> %d\n", fromPeg, toPeg);
   }
   else
   {
      otherPeg = theOtherPeg(fromPeg, toPeg);
      ToH(numDiscs - 1, fromPeg, otherPeg);
      printf("%d -> %d\n", fromPeg, toPeg);
      ToH(numDiscs - 1, otherPeg, toPeg);
   }
}
```

103

103

```c
#include <stdio.h>

/* This is a test program for the recursive
 * function for the Towers of Hanoi.
 */

int main(void)
{
   int n;

   printf("Enter number of discs: ");
   scanf("%d", &n);

   if (n > 0)
      ToH(n, 1, 3);
   else
      printf("Invalid n value.\n");

   return 0;
}
```

104

104

Keys to Successful Recursion

- In the recursive function, you must use a branching statement, e.g. *if-else* statement,
  - Leading to different cases/branches
  - Uses the input parameter of the function
- One branch should include a recursive call of the function
  - The recursive call solve a "smaller" version of the task performed by the function
- One branch must include no recursive call. This is the *base case*

105

Iteration vs. Recursion

- Any problem that can be solved recursively can also be solved iteratively (non-recursively)

Iteration	Recursion
• Iterative uses a repetition structure – e.g. for loop	• Recursion uses a selection structure – e.g. if-else
• Involve repetition through the explicit use of a repetition structure	• Involve repetition through repeated function calls
• Involve a termination test – terminates when the loop-continuation condition fails	• Involve a termination test – terminates when a base case is reached
• Approach termination by modifying a counter until the counter reaches a value that makes the loop-continuation condition fail	• Approach termination by producing simpler versions of the original problem until base case is reached

106

106