Tutorial

Programming & Data Structure: CS 11001

Section - 4/D

Department of Computer Science and Engineering

I.I.T. Kharagpur

Spring Semester: 2013 - 2014 (27.02.2014)

Download

Download the file tut270214.pdf from Programming & Data Structures ... of

http://cse.iitkgp.ac.in/~goutam

View the file using the command acroread & or xpdf &

Definition of $n!$

$$n! = \begin{cases} 1 & \text{if } n = 0, \\ n * (n-1)! & \text{if } n > 0 \end{cases}$$

Computation of $4!$

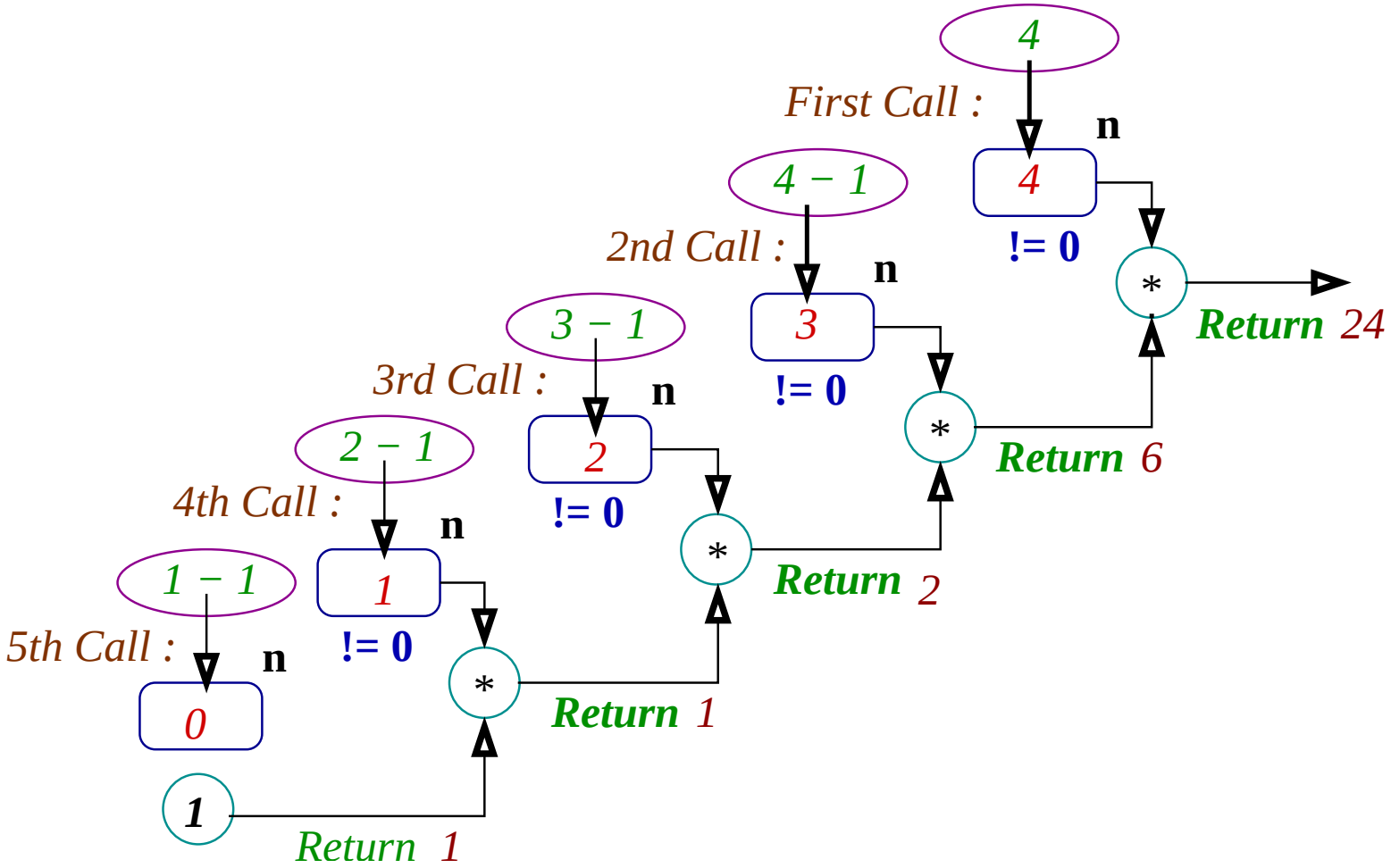
$$
\begin{aligned}
4! &= 4 \times 3! \\
&= 4 \times (3 \times 2!) \\
&= 4 \times (3 \times (2 \times 1!)) \\
&= 4 \times (3 \times (2 \times (1 \times 0!))) \\
&= 4 \times (3 \times (2 \times (1 \times 1))) \\
&= 4 \times (3 \times (2 \times 1)) \\
&= 4 \times (3 \times 2) \\
&= 4 \times 6 = 24
\end{aligned}
$$

Recursive Computation

```
int factorial(int n){ // factorialFR1.c
    if(n == 0) return 1 ;
    return n*factorial(n-1);
}
```

Different Calls and Incarnations of n

Actual Parameter is 4


4
First Call :
n
4
!= 0
4 − 1
2nd Call :
n
3
!= 0
3 − 1
3rd Call :
n
2
!= 0
2 − 1
4th Call :
n
1
!= 0
1 − 1
5th Call :
n
0
1
Return 1
*
Return 1
*
Return 2
*
Return 6
*
Return 24

Tutorial VII.1

Write a recursive function `int addDig(int n)` to add the digits of an integer `n`.

Tutorial VII.2

Write a recursive function
`void printDig(int n)` to print the digits of a positive integer `n`.

Rational as Continued Fraction

A *rational number*, $\frac{p}{q}$, where p, q are non-negative integers and $q \neq 0$, may be expressed as a **simple continued fraction**. Following is an example of $\frac{\mathbf{61}}{\mathbf{27}}$.

$$\frac{61}{27} = 2 + \cfrac{1}{3 + \cfrac{1}{1 + \cfrac{1}{6}}}$$

This may be represented in a compact form as $[2, 3, 1, 6]$. Similarly, $\frac{\mathbf{27}}{\mathbf{61}}$ can be represented as $[0, 2, 3, 1, 6]$.

Tutorial VII.3

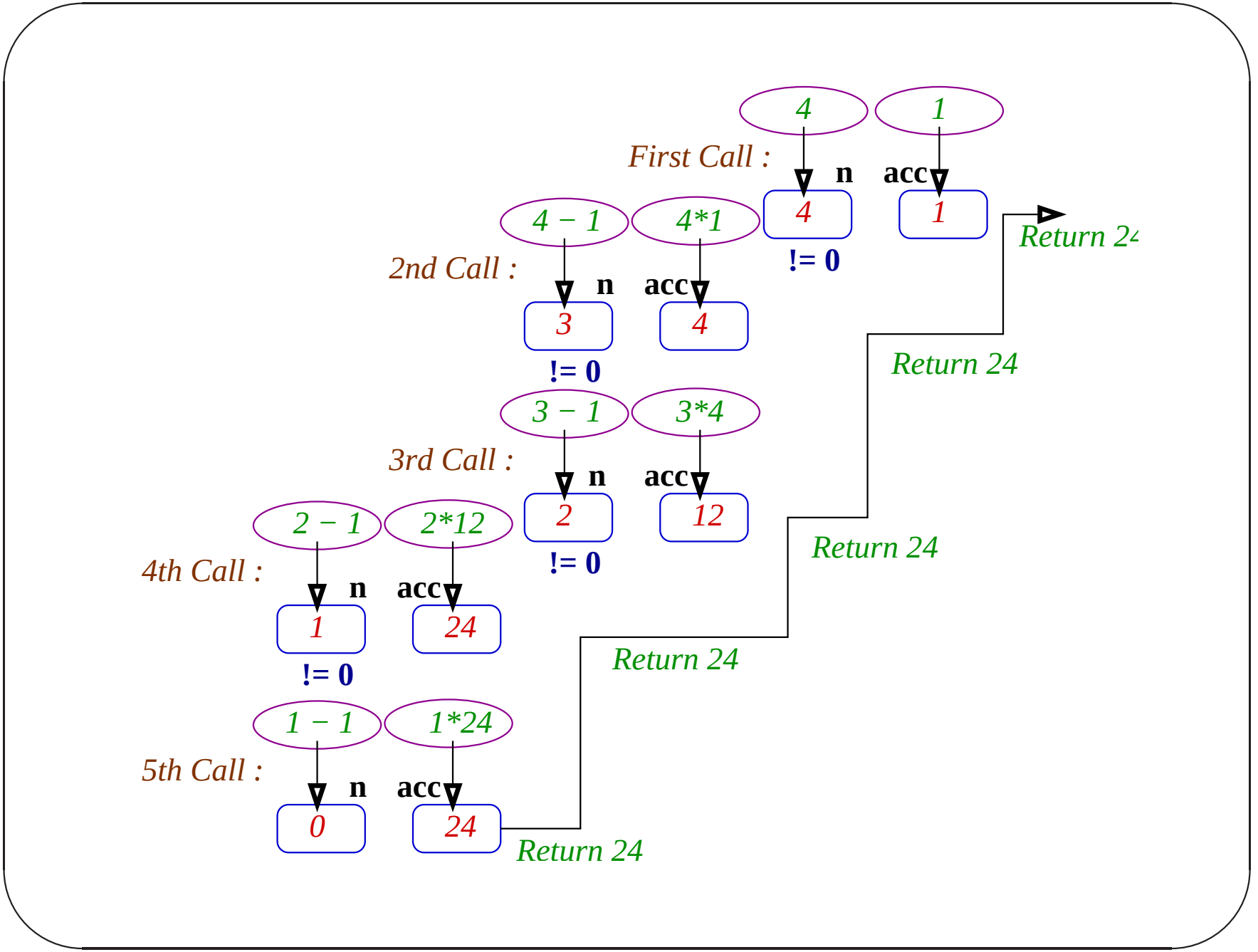
Write a recursive function `void contFrac(int den, int num)` to print the continued fraction corresponding to $\frac{\texttt{num}}{\texttt{den}}$.

Another Factorial

Following function computes $n!$ when invoked as `fact(n, 1)`.

```
int factIR(int n, int acc) {  // factorialFR2.c
    if (n == 0) return acc ;
    return factIR(n-1, n*acc) ;
}
```

Computation of `factIR(4,1)`


4
1
First Call :
n
acc
4
1
4 − 1
4*1
Return 24
2nd Call :
!= 0
n
acc
3
4
Return 24
!= 0
3 − 1
3*4
3rd Call :
n
acc
2
12
2 − 1
2*12
Return 24
!= 0
4th Call :
n
acc
1
24
Return 24
!= 0
1 − 1
1*24
5th Call :
n
acc
0
24
Return 24

Definition of Fibonacci Number

$$f(n) = \begin{cases} 0 & \text{if } n = 0, \\ 1 & \text{if } n = 1, \\ f(n-1) + f(n-2) & \text{if } n > 1 \end{cases}$$

Fibonacci Iterative

```
int fibI(int n){     // fibIR1.c
    int f0=0, f1=1, i;
    if(n < 2) return n ;
    for(i=2; i<=n; ++i) {
          f1 += f0;
          f0 = f1 - f0;
    }
    return f1 ;
}
```

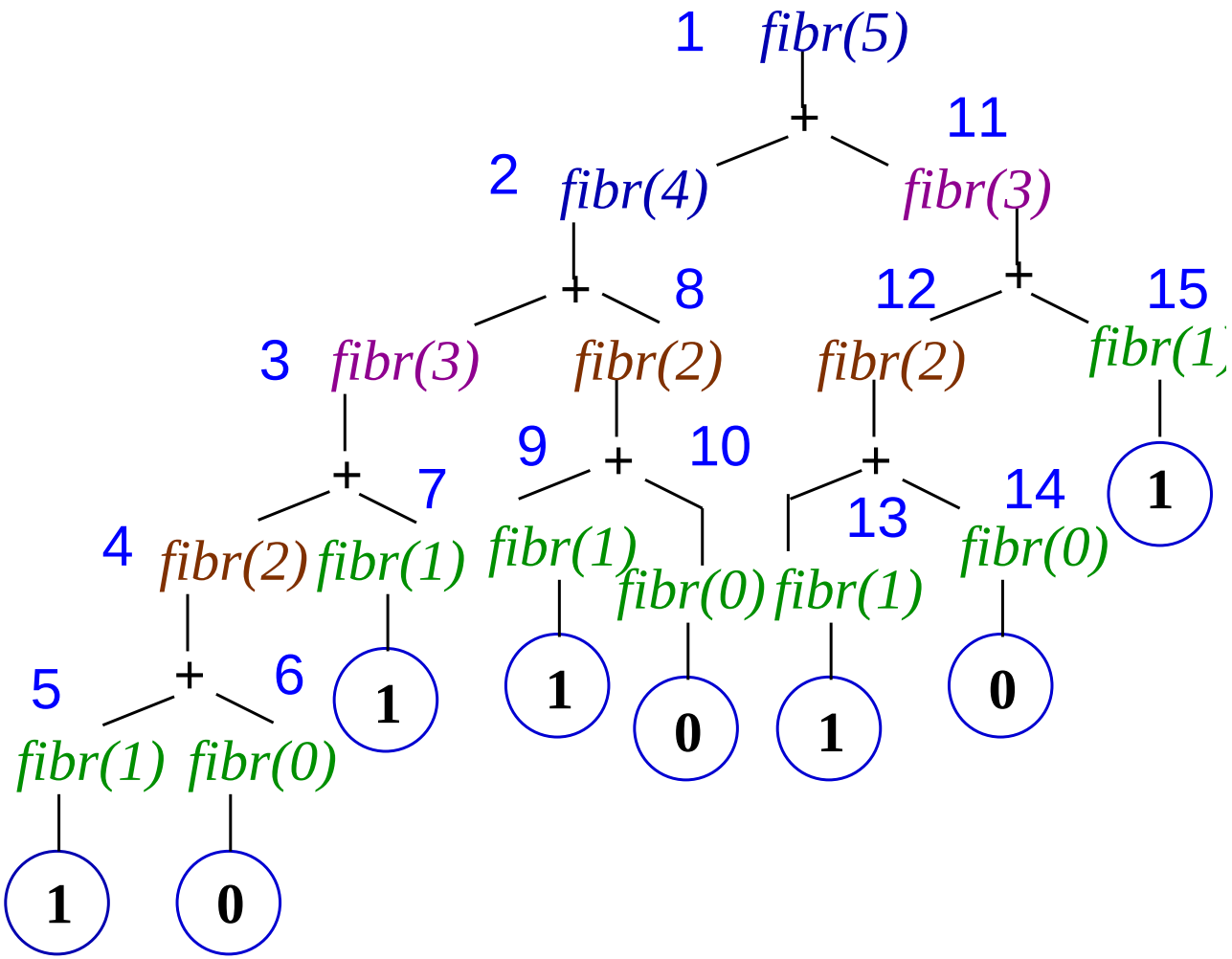
Fibonacci Recursive

```
int fibR(int n){        // fibIR1.c
    if(n < 2) return n;
    return fibR(n-1)+fibR(n-2);
}
```

Tutorial VII.4

Which implementation of Fibonacci number computation (`fibI()` or `fibR()`) is more efficient? Explain the reason.

Call Tree `fibR(5)`



Tutorial VII.5

Write a recursive function `int fibIR()` so that it will be efficient (similar to `factIR(int n, int acc)`).