

# Rekursif

PEMOGRAMAN DASAR C++

# Definisi Rekursif

- ▶ Method yang memanggil dirinya sendiri
- ▶ Lebih memudahkan pemecahan masalah tertentu
- ▶ Memerlukan kondisi penghentian (*stopping condition* atau *base case*)
- ▶ Contoh:  
Faktorial  
 $0! = 1;$   
 $n! = n \times (n - 1)!; n > 0$

# Deklarasi Rekursif

- ▶ Contoh deklarasi rekursif:

```
public static long factorial(int n)
{
    if (n==0) // stopping condition / base case
        return 1;
    else
        return n * factorial(n-1); // recursive
    call
}
```

- ▶ Pemanggilan factorial (diri sendiri) → rekursif
- ▶ Dipanggil sampai mencapai kondisi  $n == 0$   
(*stopping condition* atau *base case*)

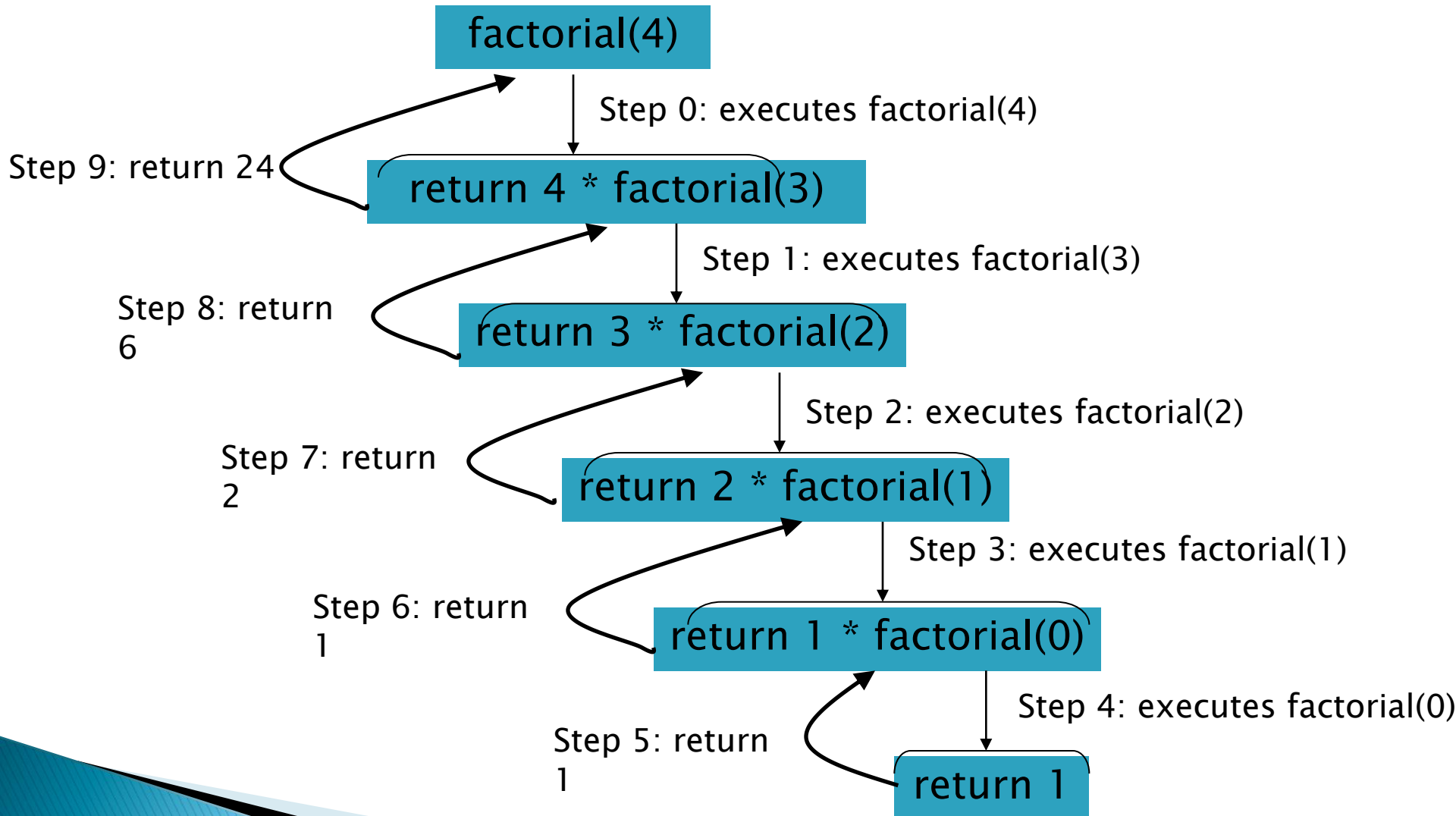
# Penggunaan Rekursif 1

```
public class Faktorial
{
    public static long faktorial(int n)
    {
        if(n==0)
            return 1;
        else
            return n * faktorial(n-1);
    }

    public static void main(String[] args)
    {
        for(int i=0; i<11; i++)
        {
            System.out.println("Hasil faktorial "+i+" adalah "+faktorial(i));
        }
    }
}
```

```
Hasil faktorial 0 adalah 1
Hasil faktorial 1 adalah 1
Hasil faktorial 2 adalah 2
Hasil faktorial 3 adalah 6
Hasil faktorial 4 adalah 24
Hasil faktorial 5 adalah 120
Hasil faktorial 6 adalah 720
Hasil faktorial 7 adalah 5040
Hasil faktorial 8 adalah 40320
Hasil faktorial 9 adalah 362880
Hasil faktorial 10 adalah 3628800
```

# Penjelasan Penggunaan Rekursif 1



# Penggunaan Rekursif 2

- ▶ Faktorial lebih baik menggunakan loop

```
int hasil = 1;
```

```
if(bil>1)
```

```
{
```

```
    for(int i=2; i<=bil; i++)
```

```
        hasil *= i;
```

```
}
```

- ▶ Faktorial baik dalam merepresentasikan rekursif

# Penggunaan Rekursif 3

```
#include<stdio.h>
void rekursi(int n);
main()
{ int x=3;
  rekursi(x); /* Pemanggilan Fungsi */
}
void rekursi(int n)
{ static int j=0;
  if(n<=0) return; /* Kondisi Perhentian pemanggilan fungsi kembali */
  printf("rekursi ke-%d\n",++j);
  rekursi(n-1); /* Pemanggilan fungsi rekursi di dalam fungsi rekursi */
}
```

## Penggunaan Rekursif 2

D:\BAHAN NGAJAR KULIAH\MATERI PENGAJARAN

```
rekursi ke-1
rekursi ke-2
rekursi ke-3
_
```

# Penjelasan Penggunaan Rekursif 3

Dalam membuat fungsi rekursi harus ditentukan kondisi perhentian. Pada contoh listing program 3 di atas kondisi perhentian adalah jika nilai  $n$  sudah lebih kecil atau sama dengan 0. Setiap kali fungsi memanggil dirinya sendiri, nilai dari  $n$  dikurangi dengan nilai 1, sehingga nilai  $n$  akhirnya akan menjadi nol dan proses rekursi akan diakhiri, sehingga fungsi ini akan memanggil dirinya sendiri sebanyak  $n$  kali.



# Penggunaan Rekursif

- ▶ Fibonacci:

Index	0	1	2	3	4	5	6	7	8	9	10	11
Seri	0	1	1	2	3	5	8	13	21	34	55	89

- ▶ Dimulai dari 0 dan 1
- ▶ Bilangan berikutnya penjumlahan dari 2 bilangan sebelumnya
- ▶  $\text{fib}(0) = 0$   
 $\text{fib}(1) = 1$   
 $\text{fib}(\text{index}) = \text{fib}(\text{index} - 2) + \text{fib}(\text{index} - 1);$   
 $\text{index} \geq 2$

# Penggunaan Rekursif

```
public class Fibonacci
{
    public static long fib(int n)
    {
        if(n<0)
            return 0;
        else
            if(n<2)
                return n;
            else
                return fib(n-1)+fib(n-2);
    }

    public static void main(String[] args)
    {
        for(int i=0; i<11; i++)
        {
            System.out.println("Bilangan fibonacci ke-"+i+" adalah "+fib(i));
        }
    }
}
```

---

# Penggunaan Rekursif

```
Bilangan fibonacci ke-0 adalah 0
Bilangan fibonacci ke-1 adalah 1
Bilangan fibonacci ke-2 adalah 1
Bilangan fibonacci ke-3 adalah 2
Bilangan fibonacci ke-4 adalah 3
Bilangan fibonacci ke-5 adalah 5
Bilangan fibonacci ke-6 adalah 8
Bilangan fibonacci ke-7 adalah 13
Bilangan fibonacci ke-8 adalah 21
Bilangan fibonacci ke-9 adalah 34
Bilangan fibonacci ke-10 adalah 55
```

# Rekursif vs Iteratif

## ▶ Rekursif:

- Bentuk alternatif kontrol program
- Repetisi tanpa loop
- Memanggil methodnya sendiri
- Memerlukan pernyataan seleksi (if) untuk *base case/stopping condition*
- Memerlukan memori lebih banyak
- Waktu eksekusi lebih lambat
- Terkadang lebih jelas, sederhana dibandingkan iteratif

## ▶ Iteratif:

- Merupakan struktur kontrol (fundamental dari bahasa pemrograman)
- Spesifikasi *loop body*
- Dikontrol oleh *loop-control-structure*
- Memori lebih sedikit dan waktu proses lebih cepat

## ▶ Permasalahan yang dapat diselesaikan oleh rekursif pasti bisa diselesaikan oleh iteratif

# Did You Know?

- ▶ *Infinite recursion* (rekursif tak terhingga) terjadi jika tidak ada *stopping condition* atau *base case*

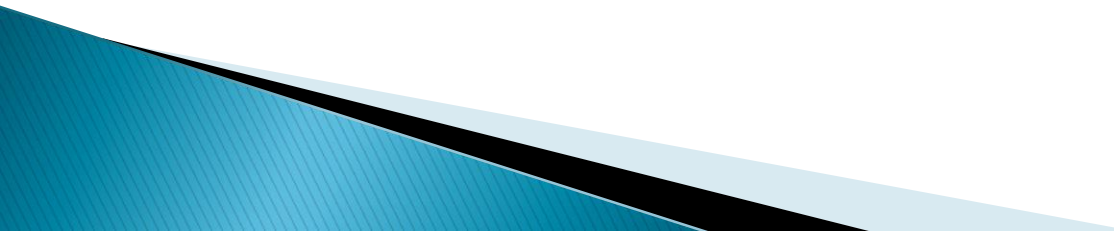
- ▶ Contoh pada Faktorial

```
public static long factorial(int n)
{
    return n * factorial(n-1); // recursive
    call
}
```

- ▶ Menyebabkan StackOverflowError
- ▶ Solusi

```
if (n==0) // stopping condition / base case
    return 1;
```

# Advanced Learning

- ▶ Tower of Hanoi: contoh rekursif klasik
  - ▶ Mudah diselesaikan dengan rekursif
  - ▶ Terdapat 3 menara (*tower*) dan beberapa cakram (*disk*) dengan ukuran berbeda
  - ▶ *Disk* atas harus lebih besar daripada *disk* bawah
  - ▶ Semua *disk* berawal dari *tower* pertama
  - ▶ Hanya 1 *disk* yang boleh dipindahkan dalam sekali waktu
  - ▶ Semua *disk* harus dipindahkan ke *tower* akhir
- 

# Advanced Learning

Source

Intermediate

Destination

Number of plates:

Move from stand 1 to stand 3

Moves till now : 52

Next

Exit

# Advanced Learning

```
import java.util.Scanner;

public class TowersOfHanoi
{
    public static void moveDisks(int n, char fromTower, char
toTower, char auxTower)
    {
        if(n==1)
            System.out.println("Move disk "+n+" from "+
fromTower+" to "+toTower);
        else
        {
            moveDisks(n-1, fromTower, auxTower, toTower);
            System.out.println("Move disk "+n+" from "+
fromTower+" to "+toTower);
            moveDisks(n-1, auxTower, toTower, fromTower);
        }
    }

    public static void main(String[] args)
    {
        Scanner input = new Scanner(System.in);
        int n;

        System.out.print("Enter number of disks : ");
        n = input.nextInt();
        System.out.println("The moves are:");
        moveDisks(n, 'A', 'B', 'C');
    }
}
```



# Advanced Learning

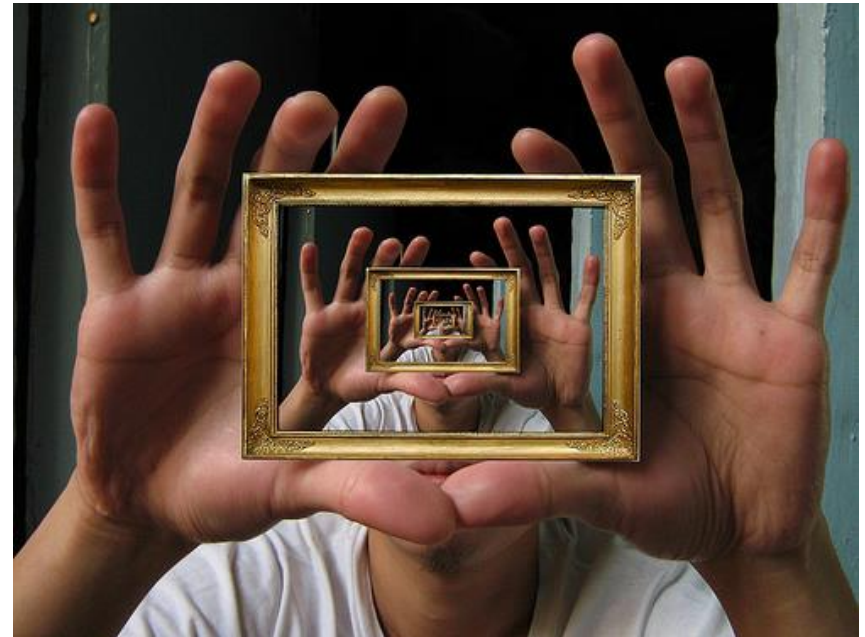
```
Enter number of disks : 3
The moves are:
Move disk 1 from A to B
Move disk 2 from A to C
Move disk 1 from B to C
Move disk 3 from A to B
Move disk 1 from C to A
Move disk 2 from C to B
Move disk 1 from A to B
```

# Fungsi Rekursif (Berulang)

Rekursif VS Iterasi

# Definisi

- ▶ Recursive = Berulang
- ▶ Recursive function = fungsi rekursif = fungsi yang berulang
- ▶ di dalam fungsi tersebut dia memanggil dirinya sendiri



# Fungsi Rekursif

- ▶ Fungsi rekursif:  
Di dalamnya terdapat pernyataan yang memanggil dirinya sendiri.
- ▶ Berguna untuk memecahkan masalah yang dapat didefinisikan secara rekursif pula.
- ▶  $n$  faktorial atau  $n! = n * n-1 * n-2 * \dots * 3 * 2 * 1$ , dan didefinisikan bahwa  $0! = 1$
- ▶ Contoh:
  - $3! = 3 * 2 * 1$
  - $4! = 4 * 3 * 2 * 1$
  - $4! = 4 * 3!$
  - $n! = n * (n-1)!$

**Faktorial (n)** atau  $n!$  didefinisikan sebagai berikut :

*jika  $n = 0$ ,  $n! = 1$*

*jika  $n > 0$ ,  $n! = n * (n-1)!$*

$$4! = 4 * 3!$$

$$3! = 3 * 2!$$

$$2! = 2 * 1!$$

$$1! = 1 * 0!$$

$$0! = 1$$

Jadi:  $4! = 4 * 3 * 2 * 1 = 24$

# Fungsi Iteratif

$$4! = 4*3! = 4*3*2! = 4*3*2*1! = 4*3*2*1 = 24$$

```
// iteratif dekremental
long faktorialIteratifDec(long n)
    mulai
        long i, faktorial = 1;
        for(i=n; i>=1; i--)
            faktorial *= i;
        return faktorial;
    selesai
tutup
```

$$4! = 1*2*3*4 = 24$$

```
// iteratif inkremental
long faktorialIteratifInc(long n){
    mulai
        long i, faktorial = 1;
        for(i=1; i<=n; i++)
            faktorial *= i;
        return faktorial;
    selesai
tutup
```

# Fungsi Rekursif

- ▶ Contoh perhitungan 5 faktorial

5!

$(5 * 4!)$

$(5 * (4 * 3!))$

$(5 * (4 * (3 * 2!)))$

$(5 * (4 * (3 * (2 * 1!))))$

$(5 * (4 * (3 * (2 * (1 * 0!)))))$

$(5 * (4 * (3 * (2 * (1 * 1)))))$

$(5 * (4 * (3 * (2 * 1))))$

$(5 * (4 * (3 * 2)))$

$(5 * (4 * 6))$

$(5 * 24)$

120

# Fungsi Rekursif

- ▶ Fungsi rekursif mempunyai dua komponen yaitu:
  - ***Base case:***  
Mengembalikan nilai tanpa melakukan pemanggilan rekursi berikutnya.  
Rekursi berakhir jika *base case* dijumpai/dipenuhi
  - ***Recursion call / Reduction step:***  
Memanggil fungsi rekursif di dalam fungsi rekursif di atas  
Menghubungkan sebuah fungsi rekursif dengan fungsi rekursif di dalamnya  
Biasanya memiliki *keyword* **return** untuk mengembalikan nilai ke fungsi yang memanggilmnya



# Fungsi Rekursif

## ► Fungsi faktorial

- Base case:  $n = 0$
- Reduction step:  $f(n) = n * f(n-1)$

```
// rekursif
long faktorialRekursif(long n)
    mulai
        if (n==0)
            return (1);
        else
            return (n * faktorialRekursif(n-1));
    selesai
tutup
```

# Rekursif vs Iteratif

## ► Contoh:

### • *Faktorial – Rekursif*

```
long faktorial(long n)
    mulai
        if(n==0)
            return (1);
        else
            return(n*faktorial(n-1));
    selesai
tutup
```



### • *Faktorial – Iteratif*

```
// dekremental
long faktorial(long n)
    mulai
        long i, faktorial = 1;
        for(i=n; i>=1; i--)
            faktorial *= i;
        return faktorial;
    selesai
tutup
```

# Rekursif vs Iteratif

## Rekursif

- Pengulangan dengan struktur seleksi (if-else) dan pemanggilan fungsi (dirinya sendiri) -> rekursi
- Pengulangan berhenti saat *base case* dijumpai/dipenuhi (konvergen terhadap base case)
- Pengulangan **tanpa henti** jika *base case* **tidak pernah** dijumpai/dipenuhi (tidak konvergen terhadap base case)
- Biaya proses lebih tinggi dengan pemanggilan banyak fungsi (butuh memori lebih besar & kerja prosesor lebih tinggi)
- Terbaca lebih jelas, model lebih dekat dengan masalah (contoh: faktorial, fibonacci)

## Iteratif

- Pengulangan dengan struktur repetisi (for/while)
- Pengulangan berhenti saat kondisi pengulangan bernilai salah (*false*)
- Pengulangan **tanpa henti** jika kondisi pengulangan selalu benar
- Biaya proses lebih rendah (kebutuhan memori lebih kecil & kerja prosesor lebih rendah) karena proses pengulangan berada dalam satu fungsi
- Terbaca kurang jelas, model kurang dekat dengan masalah (contoh: faktorial, fibonacci)

# Kekurangan Rekursi

- ▶ Meskipun penulisan program dengan cara rekursif bisa lebih jelas dan pendek, namun fungsi rekursif memerlukan :
  - Memori yang lebih banyak untuk mengaktifkan *stack* (memori yang digunakan untuk pemanggilan fungsi).
  - Waktu lebih lama untuk menjejaki setiap rekursi melalui *stack*.



# Kapan Rekursi?

- ▶ Secara umum, hanya jika :
  - Penyelesaian sulit dilaksanakan secara iteratif
  - Efisiensi dengan cara rekursif masih memadai
  - Efisiensi bukan masalah dibandingkan dengan kejelasan logika program
  - Tidak mempertimbangkan faktor penghematan memori dan kecepatan eksekusi program

Kecepatan kerja dan penghematan memori  
(iteratif)

VS

Perancangan logika yang baik (rekursif)

# Bilangan Fibonacci

- ▶ Urutan bilangan 0, 1, 1, 2, 3, 5, 8, 13 ... disebut bilangan Fibonacci.
- ▶ Hubungan antara satu angka dengan angka berikutnya didefinisikan secara rekursif sebagai berikut :
  - $\text{Fib}(n) = n$ , jika  $n = 0$  atau 1
  - $\text{Fib}(n) = \text{Fib}(n-2) + \text{Fib}(n-1)$ , jika  $n \geq 2$

- ▶ Misalkan jika ditanya berapa suku ke-4 dari barisan fibonacci?

- ▶  $n = 4$

$$\text{fibonacci}(4) =$$

$$= \text{fibonacci}(3) + \text{fibonacci}(2)$$

$$= (\text{fibonacci}(2) + \text{fibonacci}(1)) + (\text{fibonacci}(1) + \text{fibonacci}(0))$$

$$= ((\text{fibonacci}(1) + \text{fibonacci}(0)) + 1) + (1 + 0)$$

$$= ((1 + 0) + 1) + 1$$

$$= (1 + 1) + 1$$

$$= 2 + 1$$

$$= 3$$

# Bilangan Fibonacci

```
int Fib(int n)
    mulai
        int f;
        if (n==0)
            f = 0;
        else if (n==1)
            f = 1;
        else
            f = Fib(n-2) + Fib(n-1);
        return f;
    selesai
tutup
```

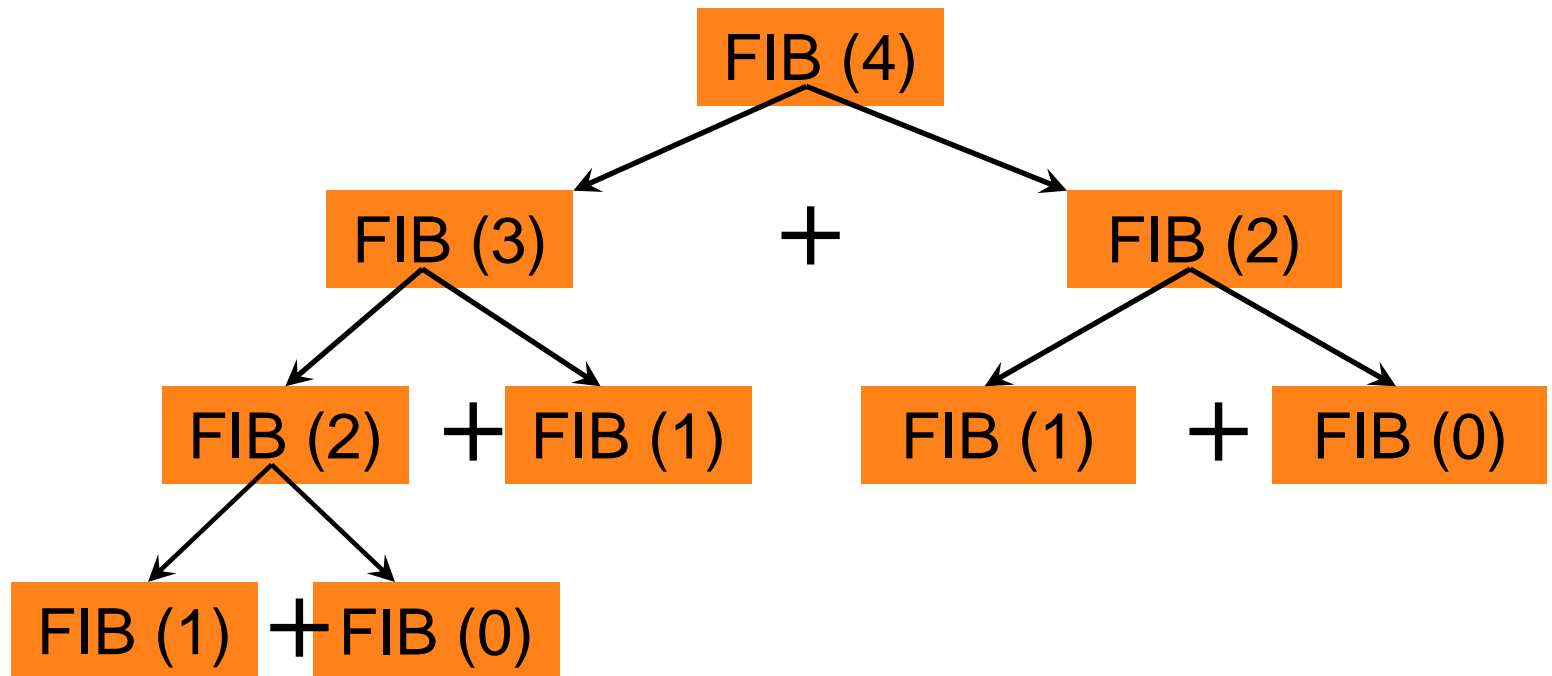
Fungsi fib() di atas ditulis secara rekursif dan disebut sebagai slow\_Fib()

Tulislah fast\_Fib() jika menggunakan iterasi.



# Bilangan Fibonacci

- ▶ Contoh : Skema fibonacci jika  $N=4$



# Latihan

- ▶ Implementasikan algoritma rekursi untuk fungsi fibonacci dan rekursif!

# Rekursif vs Iteratif

- Contoh:

- *Faktorial - Rekursif*

```
long faktorial(long n)
    mulai
        if (n==0)
            return (1);
        else
            return (n*faktorial (n-1));
    selesai
tutup
```



- *Faktorial - Iteratif*

```
// dekremental
long faktorial(long n)
    mulai
        long i, faktorial = 1;
        for(i=n; i>=1; i--)
            faktorial *= i;
        return faktorial;
    selesai
tutup
```

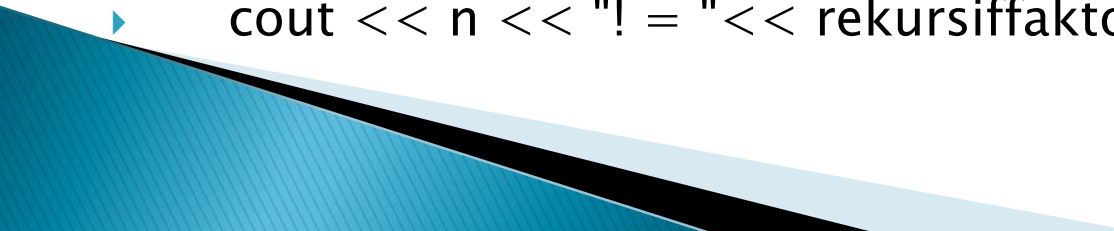
# Rekursif faktorial C++

```
1
2 long rekursiffaktorial(int f)
3
4 {
5     if (f == 0)
6         return 1;
7     else
8         return f * rekursiffaktorial(f - 1);
9 }
```

# Rekursif faktorial C++

```
▶ #include <iostream>
▶ using namespace std;
▶ long rekursiffaktorial(int f)
▶ {
▶     if (f == 0)
▶         return 1;
▶     else
▶         return f * rekursiffaktorial(f - 1);
▶ }

▶ int main()
▶ {
▶     int x;
▶
▶     int n = 4;
▶
▶     cout << n << "! = " << rekursiffaktorial(n) << endl;
```



# Rekursif faktorial C++

- ▶ `n = 9;`
- ▶ `cout << n << "! = " << rekursiffaktorial(n) << endl;`
- ▶ `cout<<"Masukan Angka yang akan difaktorialkan : ";`
- ▶ `cin>>x;`
- ▶ `cout << x <<"! = " << rekursiffaktorial(x) <<endl;`
- ▶
- ▶ `return 0;`
- ▶ `}`

## Algoritma

Judul : Algoritma Pangkat  
secara rekursif

Kamus :

X, Y : Integer

Algoritma:

{fungsi pangkat secara  
rekursif}

FUNCTION

Pangkat(X:Integer,Y:Integer)

Integer

IF Y = 0 THEN

Pangkat  $\leftarrow$  1

ELSE

Pangkat  $\leftarrow$  X \* Pangkat(X, Y-

1)

ENDIF

END FUNCTION

{Bagian Pemanggil}

INPUT X, Y

OUTPUT Pangkat(X, Y)

# Faktorial C++

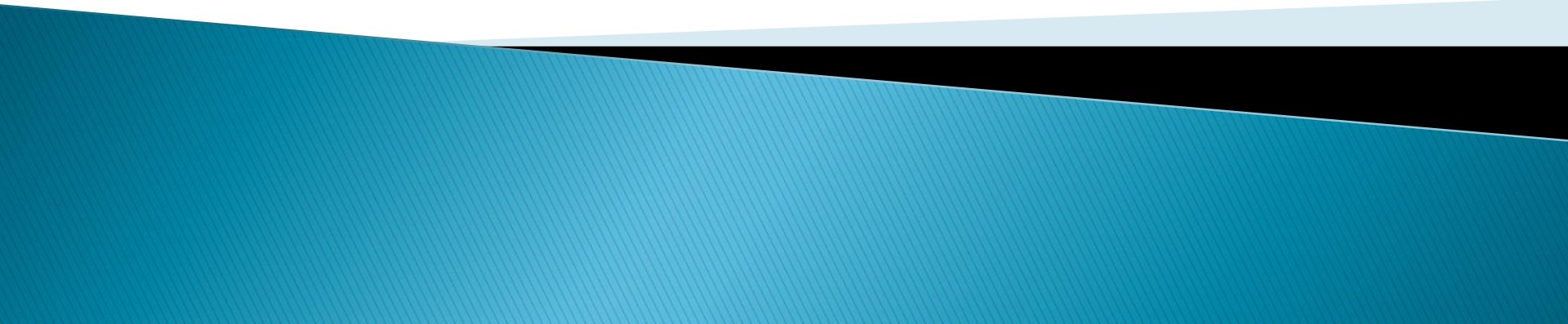
```
#include <stdio.h>
#include <conio>

int Faktorial(int n)
{
    if ((n == 0) || (n == 1))
        return (1);
    else
        return (n * Faktorial(n-1));
}

void main()
{
    int x;
    cout<<"program faktorial";
    cout<<"Masukkan berapa faktorial : ";
    cin>>n;
    cout<<"Hasil faktorial = ";
    Faktorial(x);
    getch();
}
```



# Rekursif Deret Fibonanci



# Rekursif

- ▶ Proses yang memanggil dirinya sendiri.
- ▶ Merupakan suatu fungsi atau prosedur
- ▶ Terdapat suatu kondisi untuk berhenti.

# Faktorial

- ▶ Konsep Faktorial

$$n! = n(n-1)(n-2)\dots 1$$

- ▶ Dapat diselesaikan dengan

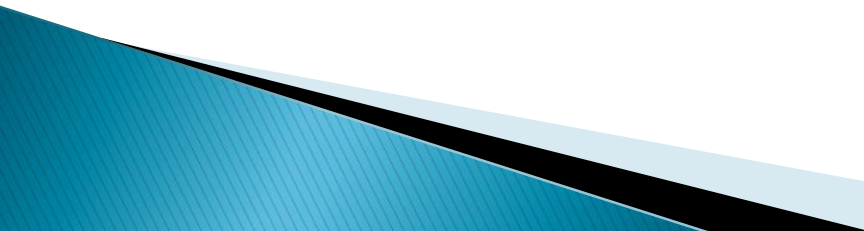
- Cara Biasa
- Rekursif

—

$$F(n) = \begin{cases} 1 & \text{jika } n=0, n=1 \\ n * F(n) & \text{jika } n > 1 \end{cases}$$

# Faktorial : Cara Biasa

```
▶ Int Faktorial(int n)
▶ {
▶     if (n<0) return -1 ;
▶     else if (n>1)
▶     {
▶         S = 1 ;
▶         for(i=2 ;i<=n;i++) S = S * n ;
▶         return S ;
▶     }
▶     else return 1 ;
▶ }
```



# Faktorial dengan Rekursif

```
▶ Int Faktorial(int n)
▶ {
    if (n<0) return -1
    else if (n>1) Return (n*Faktorial(n-1))
    Else Return 1 ;
▶ }
```

# Deret Fibonacci

- ❑ Leonardo Fibonacci berasal dari Italia 1170-1250
- ❑ Deret Fibonacci  $f_1, f_2, \dots$  didefinisikan secara rekursif sebagai berikut :
$$f_1 = 1$$
$$f_2 = 2$$
$$f_n = f_{n-1} + f_{n-2} \quad \text{for } n \geq 3$$
- ❑ Deret: 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597,...

# Deret Fibonacci

- ▶ procedure *fab*(n)
- ▶
- ▶ if  $n=1$  then  
return 1
- ▶
- ▶ if  $n=2$  then  
return 2
- ▶ return (*fab*( $n-1$ ) + *fab*( $n-2$ ))
- ▶ end

# Rekursif Tail

- ▶ Jika pernyataan terakhir yang akan dieksekusi berada dalam tubuh fungsi
- ▶ Hasil yang kembali pada fungsi tsb bukanlah bagian dari fungsi tersebut.
- ▶ Tidak memiliki aktivitas selama fase balik.



# Rekursif Tail : Faktorial()

$$F(n,a) = \begin{cases} a & \text{jika } n=0, n=1 \\ F(n-1,na) & \text{jika } n>1 \end{cases}$$

$$F(4,1) = F(3,4)$$

Fase awal

$$F(3,4) = F(2,12)$$

$$F(2,12) = F(1,24)$$

$$F(1,24) = 24$$

Kondisi Terminal

24

Fase Balik  
Rekursif Lengkap

# Latihan

## ▶ Algoritma BinRec(n)

- //input : Bilangan desimal integer positif n
- //output : Jumlah digit biner yang dinyatakan dengan n

If (n=1) return 1

Else return BinRec( $\lfloor n/2 \rfloor$ ) + 1

# Referensi

- ▶ Introduction to Java Programming. 7ed. Liang. 2009. ch 20
- ▶ Java Software Solutions. 5ed. Lewis & Loftus. 2007. ch 11
- ▶ Fibonacci.  
[http://en.literateprograms.org/Fibonacci\\_numbers\\_\(Java\)](http://en.literateprograms.org/Fibonacci_numbers_(Java))
- ▶ Palindrome. <http://en.wikipedia.org/wiki/Palindrome>
- ▶ Towers of Hanoi.
  - [http://www.codeproject.com/KB/graphics/Towers\\_of\\_Hanoi/Towers\\_of\\_Hanoi.jpg](http://www.codeproject.com/KB/graphics/Towers_of_Hanoi/Towers_of_Hanoi.jpg)
  - [http://www.codeproject.com/KB/graphics/Towers\\_of\\_Hanoi.aspx](http://www.codeproject.com/KB/graphics/Towers_of_Hanoi.aspx)
  - [http://www.java2s.com/Tutorial/Java/0100\\_\\_Class-Definition/TheTowersofHanoi.htm](http://www.java2s.com/Tutorial/Java/0100__Class-Definition/TheTowersofHanoi.htm)
  - <http://pirate.shu.edu/~wachsmut/Java/Hanoi/index.html>