

ALGORITMA dan STRUKTUR DATA



RECURSIVE FUNCTION

Contoh fungsi yang didefinisikan secara rekursif



$$f(0) = 3$$

$$f(n + 1) = 2f(n) + 3$$

Maka

$$f(0) = 3$$

$$f(1) = 2f(0) + 3 = 2 \cdot 3 + 3 = 9$$

$$f(2) = 2f(1) + 3 = 2 \cdot 9 + 3 = 21$$

$$f(3) = 2f(2) + 3 = 2 \cdot 21 + 3 = 45$$

$$f(4) = 2f(3) + 3 = 2 \cdot 45 + 3 = 93$$

Fungsi Rekursif



- Fungsi yang berisi definisi dirinya sendiri
- Fungsi yang memanggil dirinya sendiri
- Prosesnya terjadi secara berulang-ulang
- Yang perlu diperhatikan adalah “stopping role”

Plus - Minus



- +Karena program lebih singkat dan ada beberapa kasus yang lebih mudah menggunakan fungsi yang rekursif
- -Memakan memori yang lebih besar, karena setiap kali bagian dirinya dipanggil, dibutuhkan sejumlah ruang memori tambahan.
- -Mengorbankan efisiensi dan kecepatan
- -Problem: rekursi seringkali tidak bisa “berhenti” sehingga memori akan terpakai habis dan program bisa hang.
- -Program menjadi sulit dibaca
- Saran: jika memang bisa diselesaikan dengan iteratif, gunakanlah iteratif!

Bentuk Umum Fungsi Rekursif



```
return_data_type function_name(parameter_list){  
    ...  
    function_name(...);  
    ...  
}
```

Problems



- Faktorial

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

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

$$\text{Berarti } 5! = 5 \times 4!$$

- **Metode Iteratif**

Salah satu cara untuk menghitung adalah dengan menggunakan loop, yang mengalikan masing-masing bilangan dengan hasil sebelumnya. Penyelesaian dengan cara ini dinamakan iteratif, yang mana secara umum dapat didefinisikan sebagai berikut:

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

Program Iteratif



```
#include <stdio.h>
int fact_it (int n)
{
    int i,fak;
    /*******
    * Menghitung sebuah faktorial dengan proses looping *
    *****/
    temp = 1;
    for (i=1; i<=n; i++)
        fak = fak * i;
    return (fak);
}
void main()
{
    int fac;
    printf("Masukkan berapa faktorial : ");
    scanf("%d",&fac);
    printf("Hasil faktorial dari adalah : %d\n", fact_it(fac));
}
```


Faktorial Rekursif





Metode Rekursif


- Cara lain untuk menyelesaikan permasalahan di atas adalah dengan cara rekursi, dimana $n!$ adalah hasil kali dari n dengan $(n-1)!$.
- Untuk menyelesaikan $(n-1)!$ adalah sama dengan $n!$, sehingga $(n-1)!$ adalah $n-1$ dikalikan dengan $(n-2)!$, dan $(n-2)!$ adalah $n-2$ dikalikan dengan $(n-3)!$ dan seterusnya sampai dengan $n = 1$, kita menghentikan penghitungan $n!$

$$F = 5 * \text{faktorial}(4);$$

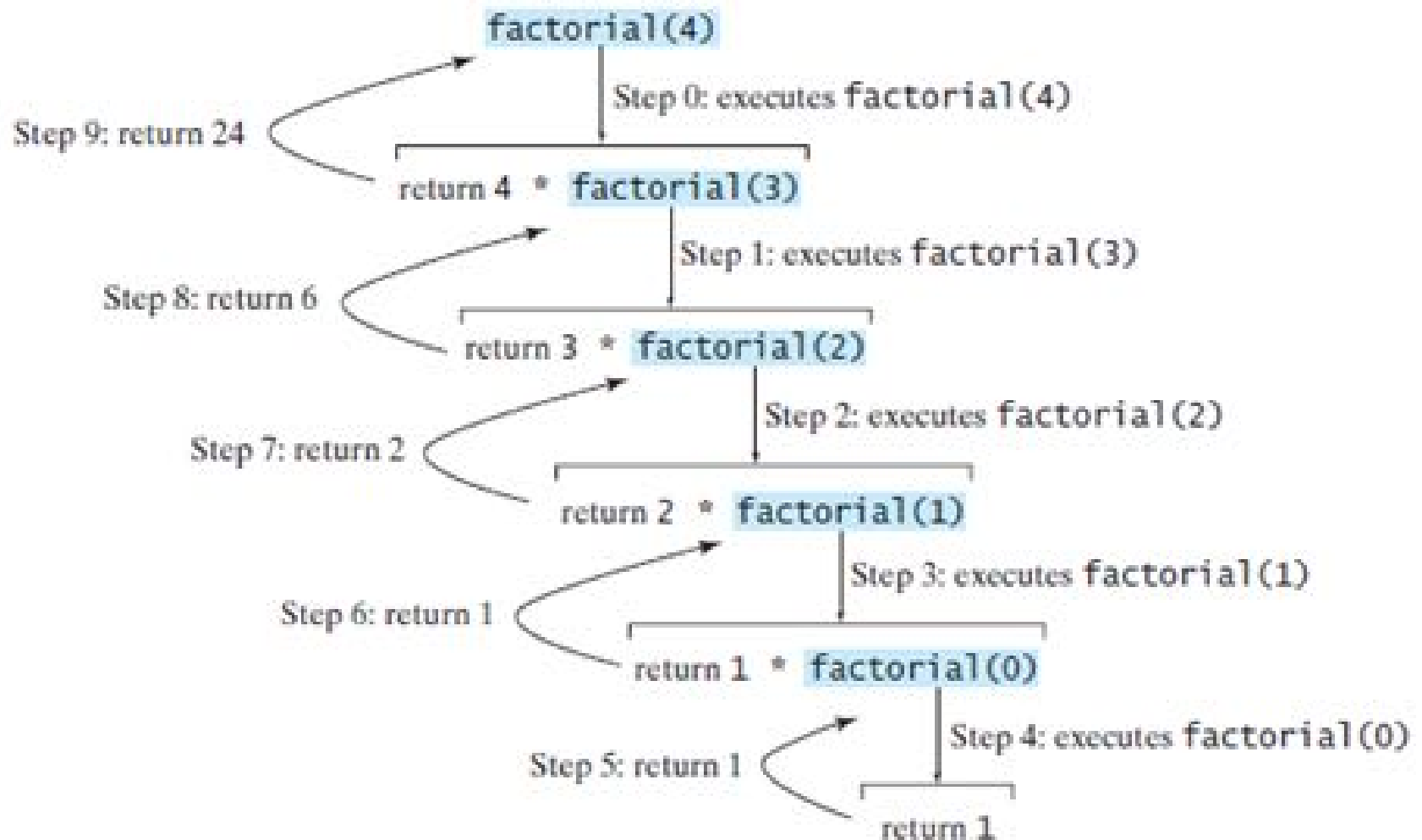

$$F = 5 * 4 * \text{faktorial}(3);$$

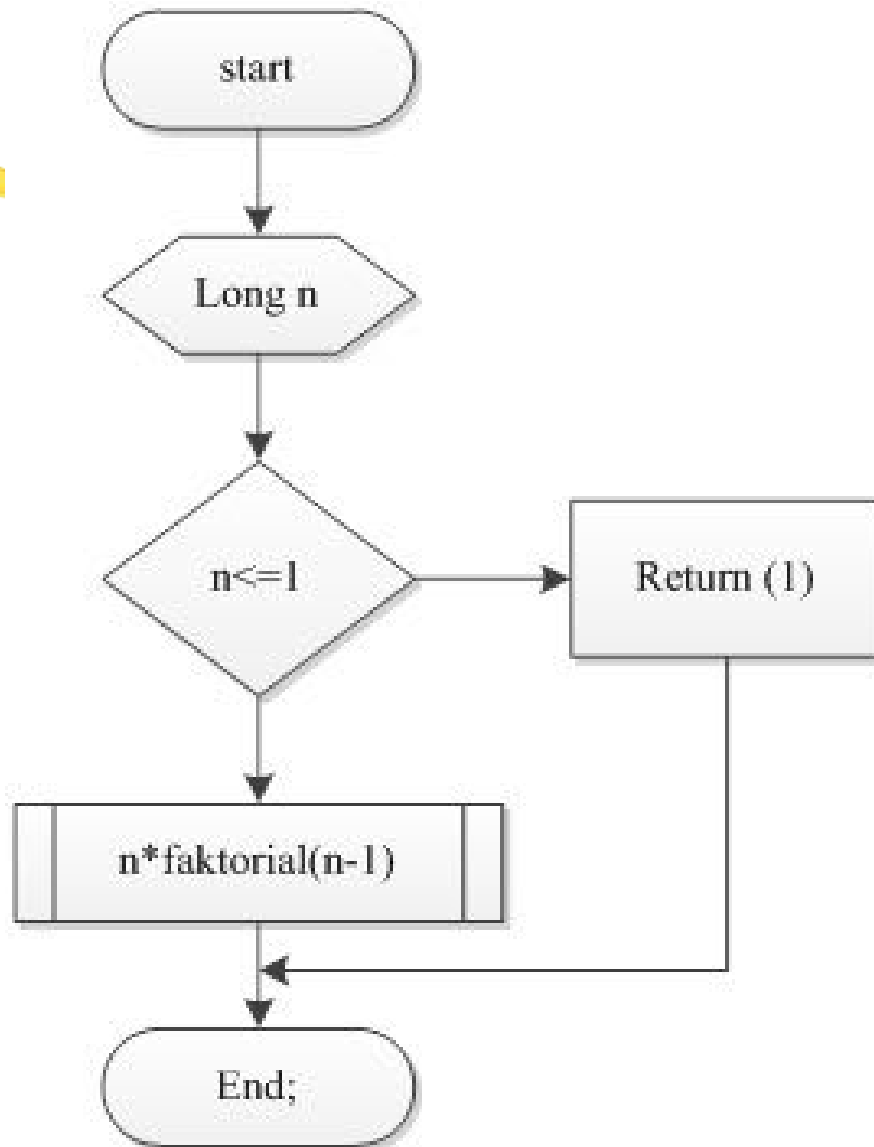

$$F = 5 * 4 * 3 * \text{faktorial}(2);$$


$$F = 5 * 4 * 3 * 2 * \text{faktorial}(1);$$


$$F = 5 * 4 * 3 * 2 * 1;$$

$$F = 120$$





Faktorial Rekursif (2)

- $n! = 1$ if $n=0$ anchor
- $n! = n*(n-1)!$ if $n>0$ inductive step
- $0! = 1$
- $1! = 1*(1-1)!$
- $= 1*0!$
- $= 1*1$
- $= 1$
- $2! = 2*(2-1)!$
- $= 2*1!$
- $= 2*1$
- $= 2$
- $3! = 3*(3-1)!$
- $= 3*2!$
- $= 3*2$
- $= 6$

Program Rekursif

```
#include <stdio.h>
int fact_rec(int n)
{
    /*******
    Menghitung sebuah faktorial secara rekursif
    *****/
    if (n < 0)
        return 0;
    else if (n == 0)
        return 1;
    else if (n == 1)
        return 1;
    else
        return n * fact_rec(n-1);
}

void main()
{
    int fac;
    printf("Masukkan berapa faktorial : ");
    scanf("%d",&fac);
    printf("Hasil faktorial dari adalah : %d\n", fact_rec(fac));
}
```

Fibonacci



- Sepasang kelinci yang baru lahir (jantan dan betina) ditempatkan pada suatu pembiakan. Setelah dua bulan pasangan kelinci tersebut melahirkan sepasang kelinci kembar (jantan dan betina). Setiap pasangan kelinci yang lahir juga akan melahirkan sepasang kelinci juga setiap 2 bulan. Berapa pasangan kelinci yang ada pada akhir bulan ke-12?

Fibo (2)

Awal bulan ke-	Jumlah pasangan		Total Pasangan
	Produktif	Blm Prod.	
1	0	1	1
2	0	1	1
3	1	1	2
4	1	2	3
5	2	3	5
6	3	5	8
7	5	8	13
8	8	13	21
9	13	21	34
10	21	34	55
11	34	55	89
12	55	89	144

Fibo (3)



- Deret Fibonacci adalah suatu deret matematika yang berasal dari penjumlahan dua bilangan sebelumnya.
- 1, 1, 2, 3, 5, 7, 12, 19, ...

Fibo Iteratif



- Secara iteratif

```
int fibonacci(int n){
    int f1=1, f2=1, fibo;
    if(n==1 || n==2) fibo=1;
    else{
        for(int i=2;i<=n;i++){
            fibo = f1 + f2;
            f1 = f2;
            f2 = fibo;
        }
    }
    return fibo;
}
```

Fibo Rekursif



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

Bilangan Fibonacci



- Untuk $N = 40$, F_N melakukan lebih dari 300 juta pemanggilan rekursif. $F_{40} = 102.334.155$
 - Berat!!!
- Aturan: Jangan membiarkan ada duplikasi proses yang mengerjakan input yang sama pada pemanggilan rekursif yang berbeda.
- Ide: simpan nilai fibonacci yang sudah dihitung dalam sebuah array

Dynamic Programming

- *Dynamic Programming* menyelesaikan sub-permasalahan dengan menyimpan hasil sebelumnya.

```
int fibo2 (int n){
    if (n <= 1) return n;
    int result[10];
    result[0] = 1;
    result[1] = 1;
    for (int ii = 2; ii <= n; ii++) {
        result[ii] = result[ii - 2]
            + result[ii - 1];
    }
    return result[n];
}
```

Tail Rekursif

- Implementasi rekursif yang lebih efficient.
- Pendekatan *Tail Recursive*.

```
public static long fib4 (int n){  
    return fiboHelp(0,1,n);  
}
```

```
static long fiboHelp(long x, long y, int n){  
  
    if (n==0) return x;  
    else if (n==1) return y;  
    else return fiboHelp(y, x+y, n-1);  
}
```

FPB (Faktor Persekutuan Terbesar)



- Misal FPB 228 dan 90:
 - $228/90 = 2$ sisa 48
 - $90/48 = 1$ sisa 42
 - $48/42 = 1$ sisa 6
 - $42/6 = 7$ sisa 0

FPB adalah hasil terakhir sebelum sisa = 0 adalah **6**

FPB (2)



- Iteratif: FPB, $m=228$ dan $n = 90$

```
do{  
    r = m % n;  
    if (r!=0){  
        m = n;  
        n = r;  
    }  
} while (r==0) ;
```

- Tampilkan n

FPB (3)



- Rekursif:

```
int FPB(int m,int n){  
    if(m==0) return n;  
    else if(m<n) return  
FPB(n,m) ;  
    else return FPB(m%n,n) ;  
}
```


Ilustrasi FPB rekursif



- FPB(228,90) $m > n$
- FPB(48,90) $m < n$
- FPB(90,48) $m > n$
- FPB(42,48) $m < n$
- FPB(48,42) $m > n$
- FPB(6,42) $m < n$
- FPB(42,6) $m > n$
- FPB(0,6) $m = 0$

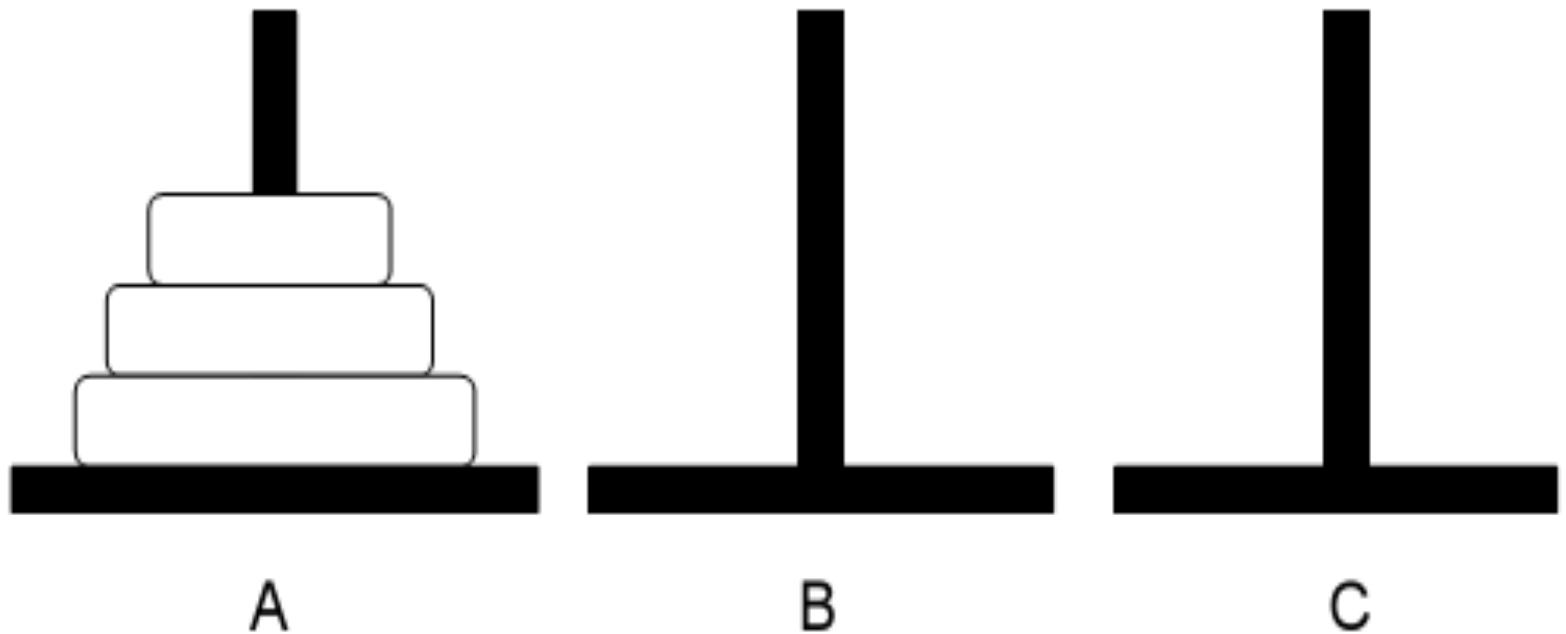
Legenda Menara Hanoi

(oleh Edouard Lucas abad 19)



- Seorang biarawan memiliki 3 menara.
- Diharuskan memindahkan 64 piringan emas.
- Diameter piringan tersebut tersusun dari ukuran kecil ke besar.
- Biarawan berusaha memindahkan semua piringan dari menara pertama ke menara ketiga tetapi harus melalui menara kedua sebagai menara tampungan.
- Kondisi:
 - Piringan tersebut hanya bisa dipindahkan satu-satu.
 - Piringan yang besar tidak bisa diletakkan di atas piringan yang lebih kecil.
- Ternyata : mungkin akan memakan waktu sangat lama (sampai dunia kiamat).
- Secara teori, diperlukan $2^{64}-1$ perpindahan. Jika kita salah memindahkan, maka jumlah perpindahan akan lebih banyak lagi.
- Jika satu perpindahan butuh 1 detik, maka total waktu yang dibutuhkan lebih dari 500 juta tahun !!.

Tower of Hanoi



Jika piringan ada 3, berapa kali jumlah perpindahan?

Tower of Hanoi



- Algoritma:
 - Jika $n == 1$, pindahkan piringan dari A ke C
 - Jika tidak:
 - Pindahkan $n-1$ piringan dari A ke B menggunakan C sebagai tampungan
 - Pindahkan $n-1$ piringan dari B ke C menggunakan A sebagai tampungan

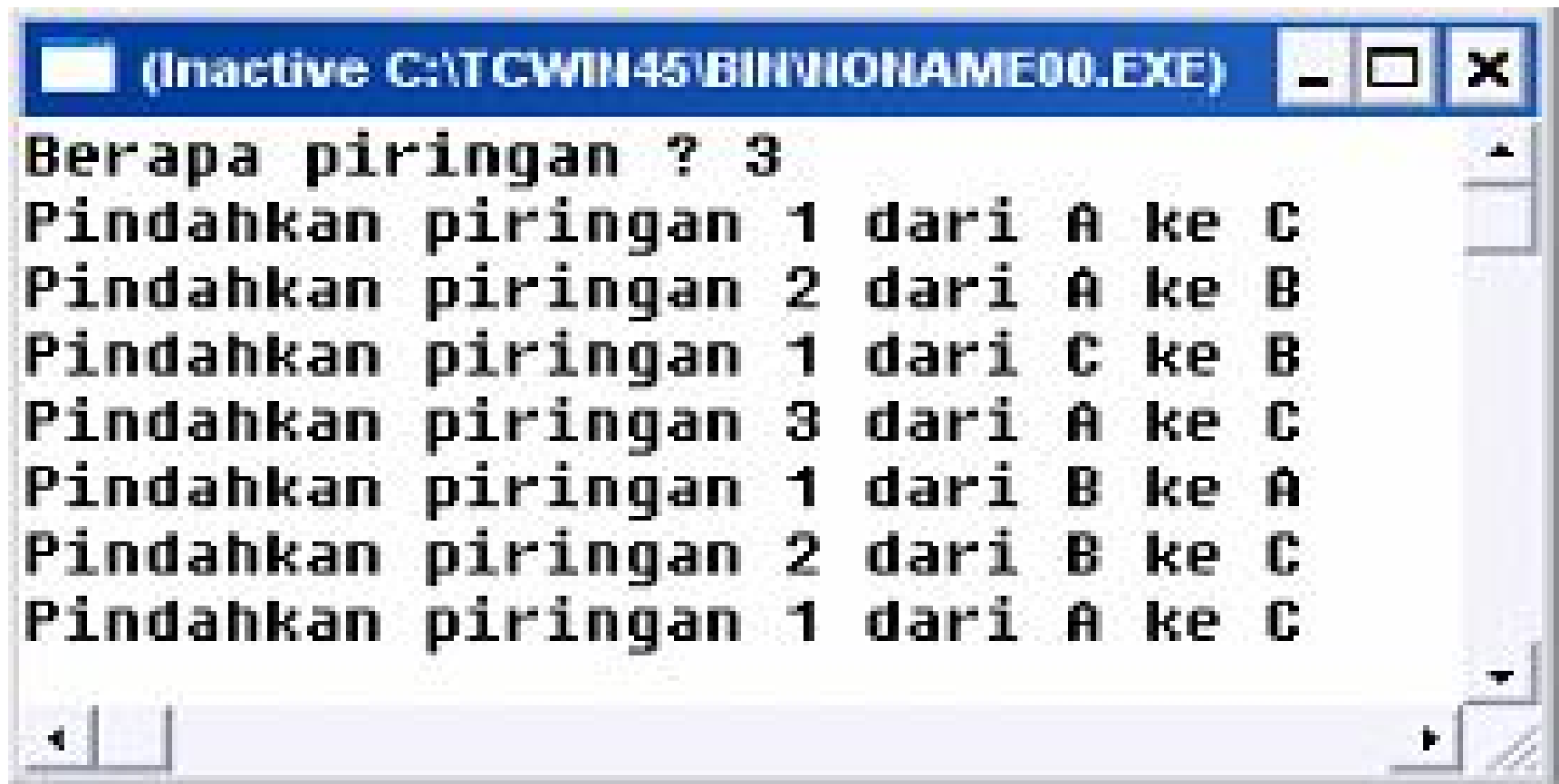
Program



```
#include <stdio.h>
void towers(int n, char awal, char akhir, char antara)
{
    if(n==1)
        printf("Pindahkan piringan 1 dari %c ke %c\n", awal,akhir);
    else{
        towers(n-1, awal, antara, akhir);
        printf("Pindahkan piringan %d dari %c ke %c\n", n, awal, akhir);
        towers(n-1, antara, akhir, awal);
    }
}

void main()
{
    int n;
    printf("Berapa piringan ? ");scanf("%d", &n);
    towers(n, 'A', 'C', 'B');
}
```

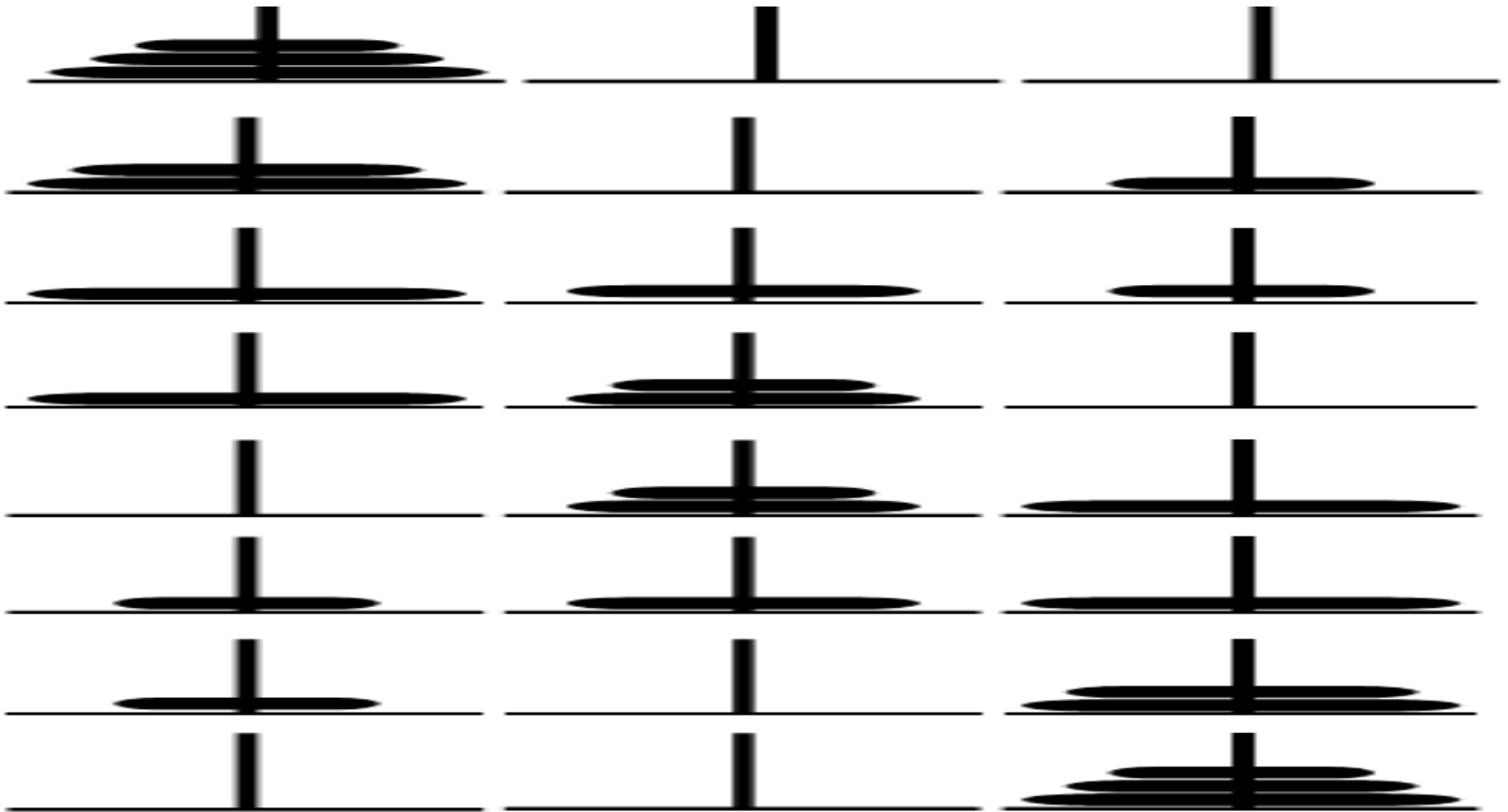
Capture Tower of Hanoi



A screenshot of a Windows command window titled "(Inactive C:\TCWIN45\BIN\IONAME00.EXE)". The window displays the solution for the Tower of Hanoi puzzle with 3 disks. The text is as follows:

```
Berapa piringan ? 3  
Pindahkan piringan 1 dari A ke C  
Pindahkan piringan 2 dari A ke B  
Pindahkan piringan 1 dari C ke B  
Pindahkan piringan 3 dari A ke C  
Pindahkan piringan 1 dari B ke A  
Pindahkan piringan 2 dari B ke C  
Pindahkan piringan 1 dari A ke C
```

Ilustrasi Tower of Hanoi



Proses Kerja

Proses:

N=3; towers(3,'A','C','B')		
towers(2,'A','B','C') Piringan 3 dari A ke C (3)	N=2; towers(2,'A','B','C')	
	towers(1,'A','C','B')	N=1; towers(1,'A','C','B')
	Piringan 2 dari A ke B (2)	Piringan 1 dari A ke C (1)
	towers(1,'C','B','A')	N=1; towers(1,'C','B','A')
		Piringan 1 dari C ke B (4)
towers(2,'B','C','A')	N=2; towers(2,'B','C','A')	
	towers(1,'B','A','C')	N=1; towers(1,'B','A','C')
	Piringan 2 dari B ke C (6)	Piringan 1 dari B ke A (5)
	towers(1,'A','C','B')	N=1; towers(1,'A','C','B')
		Piringan 1 dari A ke C (7)

Pemangkatan



```
int power2(int m,int n){  
    int p=1;  
    for(int i=1;i<=n;i++)  
        p*=m;  
    return p;  
}
```

```
int power(int m,int n){  
    if(n==1||n==0) return m;  
    else return m*power(m,n-1);  
}
```

Analisis Algoritma



- Algoritma adalah urutan langkah yang tepat dan pasti dalam memecahkan suatu masalah secara logis.
- Beberapa masalah dapat diselesaikan dengan algoritma yang bermacam-macam asal hasilnya sama.
- Setiap bahasa pemrograman memiliki kelebihan dan kekurangan dalam mengimplementasikan algoritma dan setiap pemrogram dapat mengimplementasikan suatu algoritma dengan cara yang berbeda-beda pula.
- Namun algoritma dapat dianalisis efisiensi dan kompleksitasnya.

Analisis Algoritma



- Penilaian algoritma didasarkan pada:
 - Waktu eksekusi (paling utama)
 - Penggunaan memori/sumber daya
 - Kesederhanaan dan kejelasan algoritma
- Analisis algoritma tidak mudah dilakukan secara pasti, maka hanya diambil:
 - Kondisi rata-rata (*average case*)
 - Kondisi terburuk (*worst case*)
- Waktu eksekusi dipengaruhi oleh:
 - Jenis data input
 - Jumlah data input
 - Pemilihan instruksi bahasa pemrograman

Analisis Algoritma



- Faktor-faktor yang menyulitkan analisis disebabkan oleh:
 - Implementasi instruksi oleh bahasa pemrograman yang berbeda
 - Ketergantungan algoritma terhadap jenis data
 - Ketidakjelasan algoritma yang diimplementasikan
- Langkah-langkah analisis algoritma
 - Menentukan jenis/sifat data input.
 - Mengidentifikasi *abstract operation* dari data input.

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

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

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

$$F(1,24) = 24 \quad \text{Kondisi Terminal}$$

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

Sumber



<http://lecturer.ukdw.ac.id/anton/download/TIstrukdat9.ppt>

<http://yuliana.lecturer.pens.ac.id/Struktur%20Data%20C/Teori%20ppt/Rekursif.ppt>

NEXT



- Tree dan Manipulasinya ...