POLITEKNIK STATISTIKA STIS

For Better Official Statistics

REKURSI

Farid Ridho, S.S.T., M.T., Nori Wilantika, S.S.T., M.T.I.

PEMBAHASAN

- Konsep Rekursi
- Penulisan Fungsi Rekusif
- Perbedaan antara rekursi dan iterasi serta kapan menggunakannya
- Teknik pemecahan masalah menggunakan rekursi
- Latihan

APA ITU REKURSI?

- Sub program dapat memanggil sub program lainnya
- Apabila sub program memanggil dirinya sendiri, inilah yang disebut dengan fungsi rekursif

To understand recursion, you have to understand recursion first

PENULISAN REKURSI PADA PROSEDUR ATAU FUNGSI

- Prosedur rekursif:

```
Procedure proc();
Begin
 proc();
End;
```

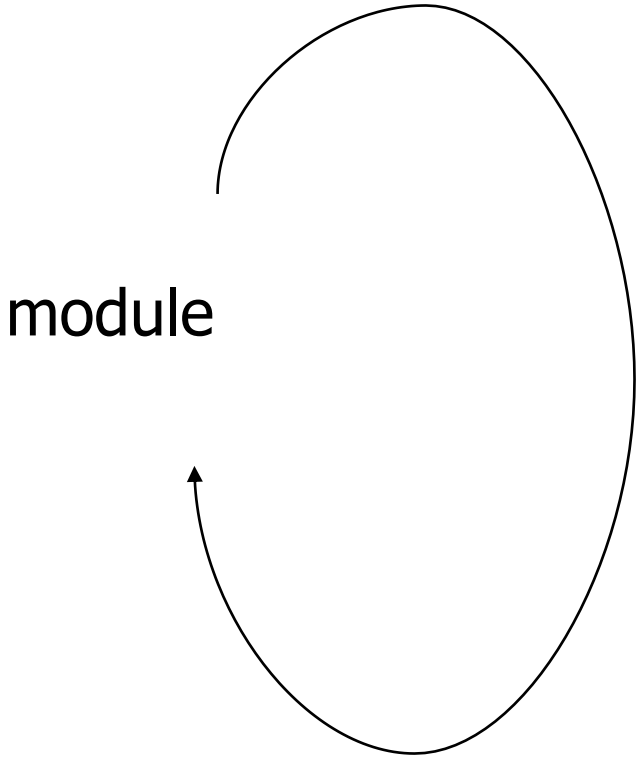
- Fungsi rekursif:

```
Function func();
Begin
 func();
End;
```

PEMANGGILAN FUNGSI REKURSIF

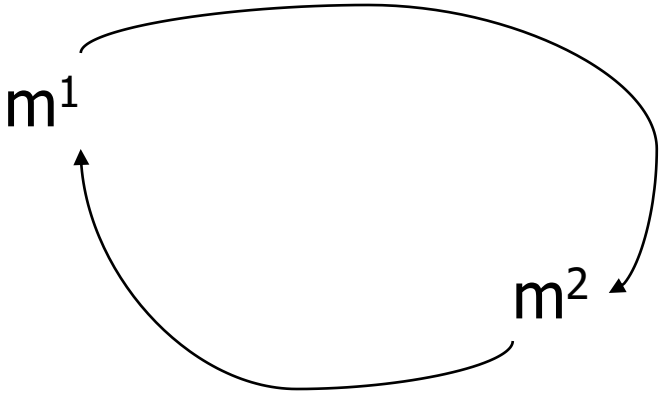
- Pemanggilan secara langsung (direct call)
- Pemanggilan tak langsung (indirect call)

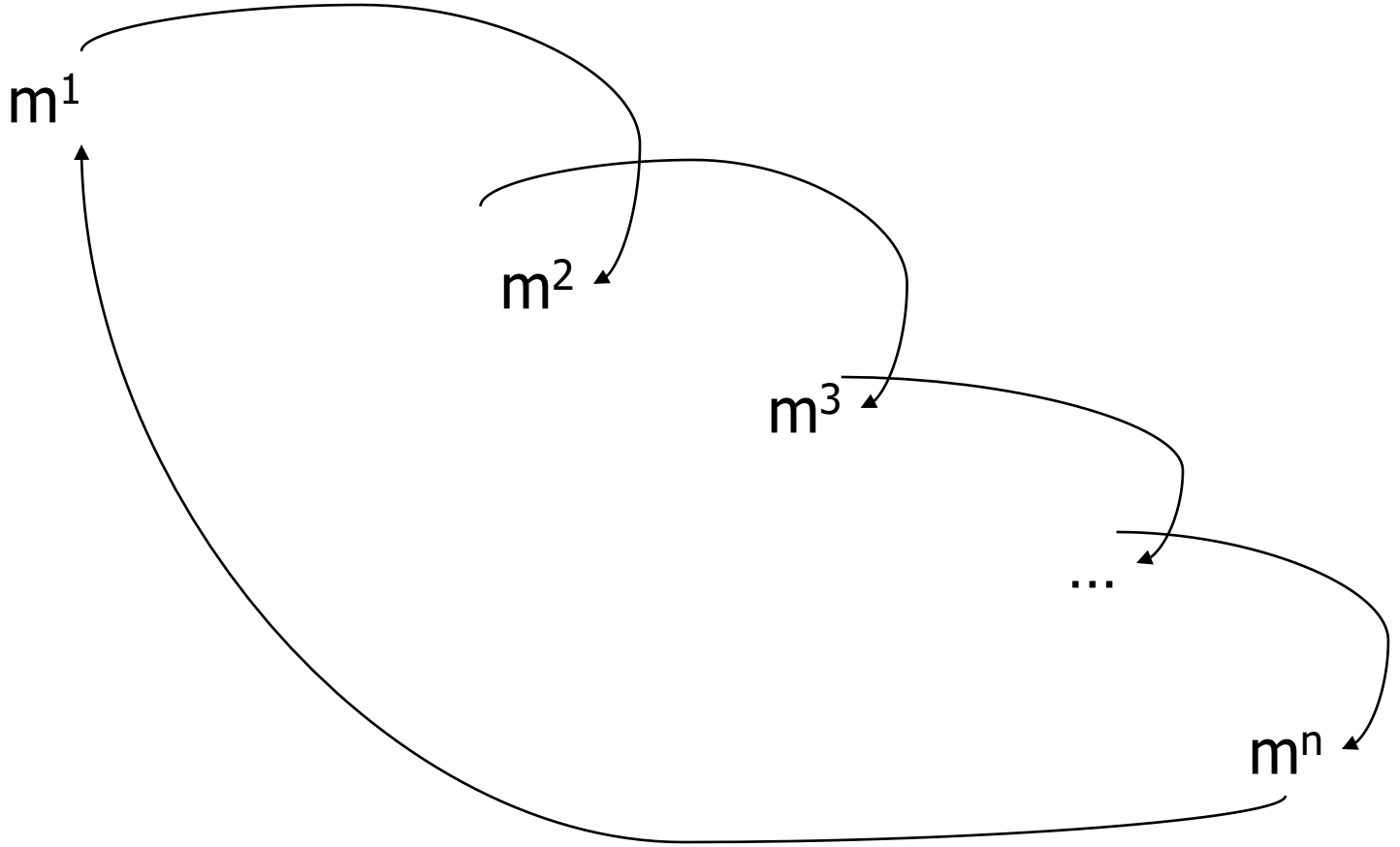
DIRECT CALL




```
procedure
proc;
begin
        :
    proc ();
        :
end;
```

INDIRECT CALL





INDIRECT CALL (2)

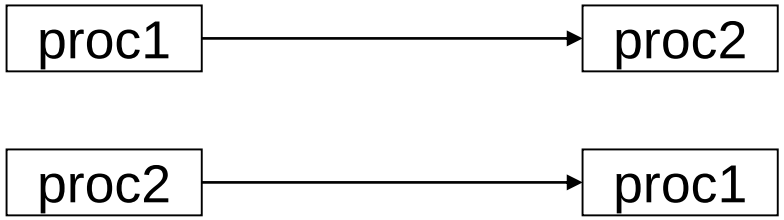
```
procedure proc1;
begin
   :
  proc2;
end;

procedure proc2;
begin
   :
  proc3;
end;

procedure proc3;
begin
   :
  proc1;
end;
```

PERMASALAHAN INDIRECT RECURSION

Contoh scenario:



Procedure mana yang harus didefinisikan lebih dulu?

PROCEDURE PROC1 PERTAMA?

```
procedure proc1;
begin
  :
 proc2;
  :
end;

procedure proc2;
begin
  :
 proc1;
  :
end;
```

PROCEDURE PROC2 PERTAMA?


```
procedure proc2;
begin
  :
 proc1;
  :
end;

procedure proc1;
begin
  :
 proc2;
  :
end;
```

What is proc1?

SOLUSI: GUNAKAN DUMMY DEFINITION

A "placeholder" for the compiler (definition comes later)

```
procedure proc2; FORWARD;
procedure proc1;
begin
  :
  proc2;
  :
end;

procedure proc2;
begin
  :
  proc1;
  :
end;
```

CONTOH 1: MENGHITUNG FAKTORIAL

Konsep Faktorial

```
n!   = n*(n-1)!
     = n*(n-1)*(n-2)!
       ...
     = n*(n-1)*(n-2)*(n-3)... 1
```

CONTOH 1: MENGHITUNG FAKTORIAL

Contoh:

Rekursif

```
faktorial(5)
   -> 5 * faktorial(4)
   -> 5 * (4 * faktorial(3))
   -> 5 * (4 * (3 * faktorial(2)))
   -> 5 * (4 * (3 * (2 * faktorial(1))))
   -> 5 * (4 * (3 * (2 * 1)))
   -> 5 * (4 * (3 * 2))
   -> 5 * (4 * 6)
   -> 5 * 24
   -> 120
```

CONTOH 1: MENGHITUNG FAKTORIAL

```
program factorialProgram;

function factorial (num :integer):integer;
var i,hasil: integer;
begin
    hasil := num;
    for i := num-1 downto 1 do
        hasil := hasil*i;
    factorial := hasil;
end;

var number, nFactorial :integer;

begin
write('Enter the number for factorial :');
readln(number);
nFactorial := factorial(number);
writeln(number, '! = ', nFactorial);
end.
```

```
program factorialProgram;

function factorial (num :integer):integer;
begin
if (num = 1) then
    factorial := 1
else
    factorial := num*factorial(num - 1);
end;

var number, nFactorial :integer;

begin
write('Enter the number for factorial :');
readln(number);
nFactorial := factorial(number);
writeln(number, '! = ', nFactorial);
end.
```

```
User Types in 3
nFactorial = factorial(3)
```

6

```
factorial (3)
if (3 = 1) then
   factorial := 1
else
   factorial := 3 * factorial(3 - 1);
```

false

2

```
factorial (2)
if (2 = 1) then
   factorial := 1
else
   factorial := 2 * factorial(2 - 1);
```

false

1

```
factorial (1)
if (1 = 1) then
   factorial := 1
else
   factorial := 1 * factorial(1 - 1);
```

true

CONTOH 2: MENGHITUNG PENJUMLAHAN

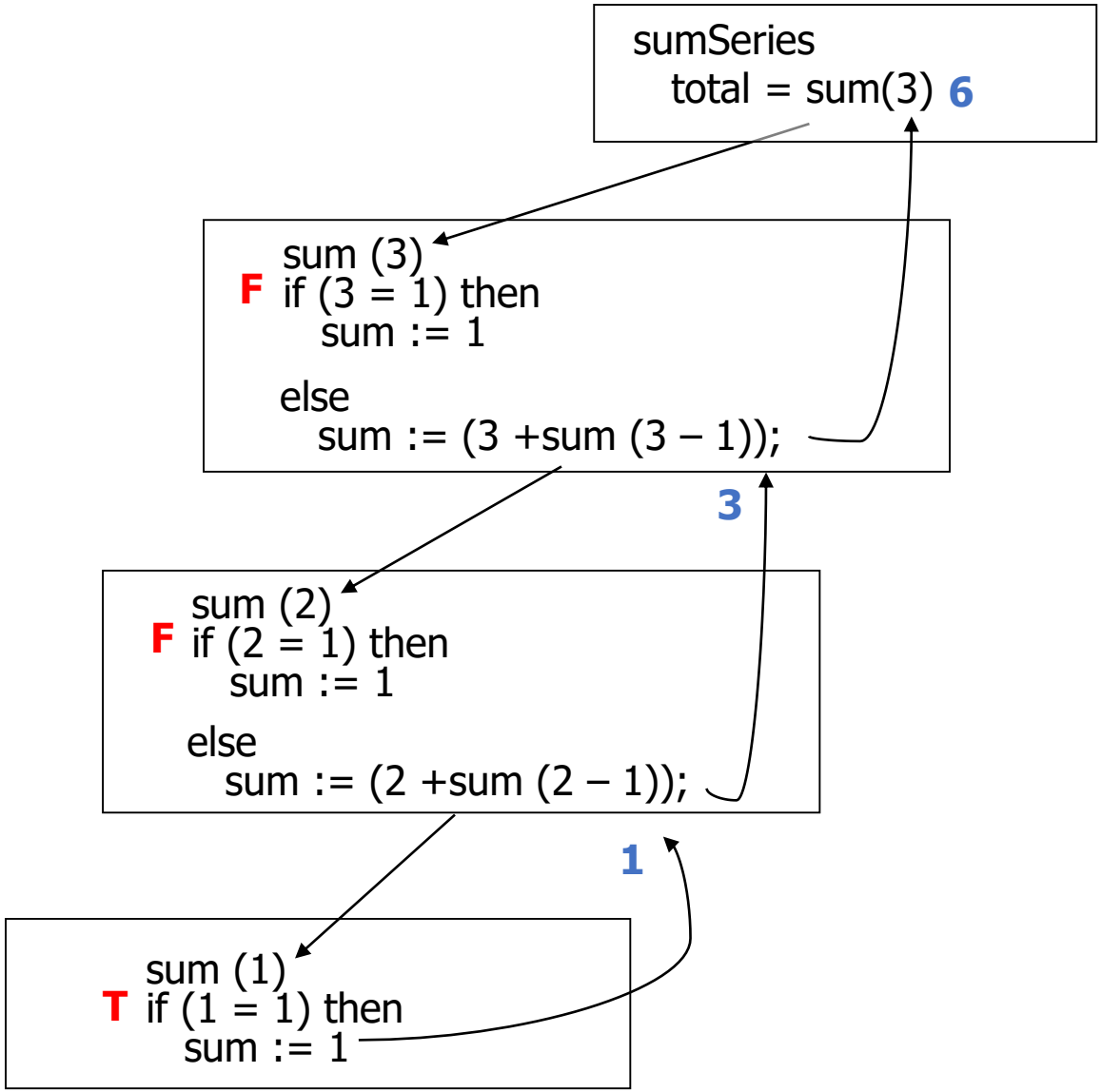
```
program sumSeries (input, output);

function sum (no : integer): integer;
begin
    if (no = 1) then
        sum := 1
    else
        sum:= (no + sum (no - 1));
end;

var lastNumber, total : integer;

begin
    write('Masukkan angka terakhir :');
    readln(lastNumber);
    total := sum(lastNumber);
    writeln('Jumlah dari 1 hingga angka tersebut ', total);
end.
```

```
function sum (no : integer): integer;
var i: integer;
begin
    if (no = 1) then
        sum := 1
    else
        sum := 0;
        for i := 1 to no do
          sum:= sum + i;
end;
```


sumSeries
total = sum(3) 6
sum (3)
F if (3 = 1) then
sum := 1
else
sum := (3 +sum (3 – 1));
3
sum (2)
F if (2 = 1) then
sum := 1
else
sum := (2 +sum (2 – 1));
1
sum (1)
T if (1 = 1) then
sum := 1

BASE CASE PADA REKURSI

- Dalam rekursi ada yang disebut sebagai *base case*,
- Dimana apabila kondisi ini tercapai maka suatu fungsi atau prosedur rekursi akan berakhir.
- Jika kondisi pada *base case* tidak tercapai (bernilai false) maka akan terjadi proses rekursi.

```
Function func();
Begin
if base_case then
 return_value
else
 func();
End;
```

PERBEDAAN REKURSI DAN ITERASI

RECURSION	ITERATIONS
Fungsi rekursif adalah fungsi yang memanggil dirinya sendiri	Iterasi adalah pengulangan dari suatu proses mengunakan struktur loop
Rekursi menggunakan struktur pemilihan	Iterasi menggunakan struktur perulangan
Rekursi tak hingga terjadi apabila langkah rekursi tidak pernah memenuhi kondisi pada base case	Perulangan tak hingga terjadi apabila kondisi perulangan tidak pernah bernilai false
Rekursi berhenti apabila kondisi (base case) dikenali	Iterasi berhenti ketika kondisi perulangan bernilai false
Rekursi lebih banyak menggunakan memory daripada iterasi	Iterasi lebih hemat memory
Rekursi tak hingga dapat menyebabkan terjadinya crash pada sistem	Perulangan tak hingga akan mengakibatkan penggunakan Siklus CPU secara berulang
Rekursi membuat kode program menjadi lebih sederhana	Iterasi membuat kode program menjadi lebih panjang

KELEBIHAN DAN KEKURANGAN REKURSI

- Kelebihan
  - solusi sangatlah efisien
  - dapat memecahkan masalah yang sulit dengan tahapan yang mudah dan singkat

- Kelemahan
  - sulit dipahami
  - perlu stack besar (stack overrun)

MEMECAHKAN MASALAH MENGGUNAKAN TEKNIK REKURSI

- Strategi yang biasa digunakan disebut dengan “Divide and Conquer”,

- yaitu dengan membagi masalah menjadi masalah dalam versi kecil kemudian menyelesaikanya.
  1. Bagi P (Problem) menjadi beberapa subproblems, P1,P2,...Pn.
  2. Selesaikan setiap subproblem sampai mendapatkan solusi S1...Sn.
  3. Gunakan S1..Sn untuk menyelesaikan masalah asal P.

CONTOH KASUS DI KEHIDUPAN SEHARI-HARI

- Amir akan memindahkan satu kardus buku yang berisi 50 buku dari dalam kantor menuju mobil.

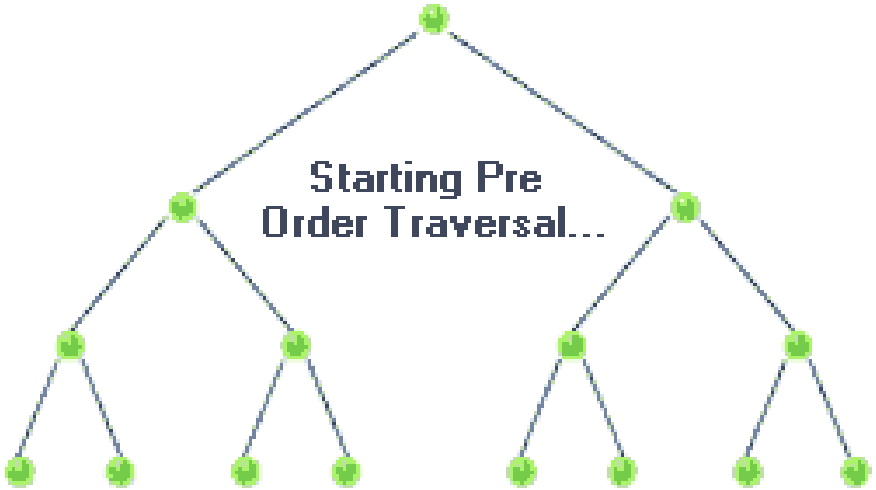
- P (Problem): memindahkan satu kardus buku yang berisi 50 buku
- Karena amir tidak mampu mengangkat keseluruhan buku maka dia membagi menjadi beberapa tumpukan P1 berisi 10 buku, P2 berisi 15 buku, P3 berisi 25 buku.
- Strategi amir dengan memindahkan setiap tumpukan buku, kemudian menyusun kembali menjadi 1 kardus buku besar di mobil ini bisa disebut sebagai rekursi.

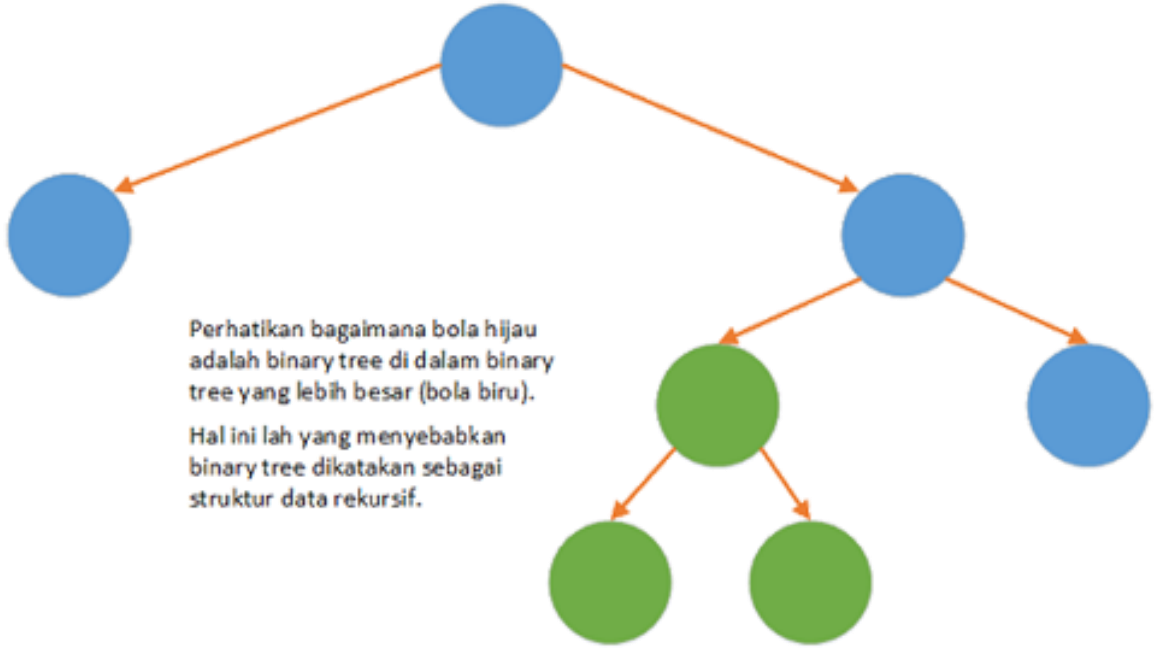
MERANCANG ALGORITMA REKURSI

- Strategi yang umum digunakan adalah “Divide and Conquer”
- Beberapa pertanyaan yang harus dijawab:
  - Bagaimana kita dapat memecah suatu masalah menjadi beberapa masalah sama dalam versi yang lebih kecil.
  - Bagaimana setiap pemanggilan fungsi dapat membuat masalah menjadi versi kecil.
  - Apa *base case* dari masalah ini?
  - Apakah algoritma akan selalu mencapai kondisi *base case*?

- Membuat fungsi dan prosedur rekursif
  - Memahami masalah yang akan dipecahkan secara tepat. Langkah ini merupakan langkah awal dari seluruh permasalahan pemrograman.
  - Tentukan seberapa besar masalah yang akan dipecahkan menjadi beberapa subprogram.
  - Kenali dan tentukan base case dari masalah yang akan dikerjakan secara tidak rekursif.
  - Terakhir kenali dan tentukan kondisi umum (general case) dengan benar

Dalam kasus tertentu kita tidak bisa menelusisri data dengan pointer seperti menggunakan i, j, k. adab kalanya data berupa tree atau pohon sehingga untuk menelusisri data perlu fungsi rekursi





Deret Fibonacci: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, . . .

```
program fibo_using_rekursif;
var x,i: integer;

function fib(n:integer):integer;
begin
    if(n=1) then fib:=1
    else if (n=2) then fib:=1
    else fib:= fib(n-1)+fib(n-2);
end;

begin
    writeln('Deret fibonacci');
    write('Jumlah Suku: ');
    readln(x);
    for i := 1 to x do
        write(fib(i),'  ');
end.
```

fib(6) = fib(4) + fib(5)
= fib(2) + fib(3) + fib(5)
= 1 + fib(3) + fib(5)
= 1 + fib(1) + fib(2) + fib(5)
= 1 + 1 + 1 + fib(3) + fib(4)
= 3 + fib(1) + fib(2) + fib(4)
= 3 + 1 + 1 + fib(2) + fib(3)
= 5 + fib(2) + fib(3)
= 5 + 1 + fib(1) + fib(2)
= 6 + 1 + 1
= 8

```
program fibo_using_iteration;

function fib(n:integer):integer;
var i,j,k,l: integer;
begin
    k:=1;
    l:=0;

      for i:=1 to n do
        begin
          write(' ',k);
          j:=k+l;
          l:=k;
          k:=j;
        end;
end;

var x: integer;
begin
    writeln('Deret fibonacci');
    write('Jumlah Suku: ');
    readln(x);
    fib(x);
end.
```

POLITEKNIK STATISTIKA STIS

For Better Official Statistics

TERIMA KASIH