# STRUKTUR DATA DAN ALGORITMA

## "Teknik Pemrograman Iteratif dan Rekursif"


Dibuat oleh:

Sirojul Munir S.Si., M.Kom /

Indra Hermawan, S.Kom, M.Kom /

Hilmy Abidzar Tawakal S.T., M.Kom /

# Outline

- Tujuan Pembelajaran
- Algoritma pemrograman iteratif
- Review fungsi
- Dasar-dasar rekursif
  - Apa itu rekursif/ recursion?
  - Aturan rekursif
- Rangkuman Materi

# Tujuan Pembelajaran

- Setelah mengikuti perkuliahan ini mahasiswa diharapkan mampu:
  - memahami algoritma pemrograman iteratif,
  - memahami algoritma pemrograman rekursif,
  - membedakan algoritma iteratif dan rekursif,
  - mengingat kembali pemrograman menggunakan fungsi
  - memahami rekursi sebagai konsep yang dapat digunakan untuk merumuskan solusi sederhana dalam sebuah permasalahan yang sulit untuk diselesaikan secara iteratif dengan menggunakan loop (FOR, WHIEL, DO..WHILE),
  - menyelesaikan suatu permasalahan dengan konsep rekursi.

# Algoritma pemrograman iteratif

- Definisi
  - Algoritma yang menggunakan teknik looping/ perulangan dalam pemecahan masalahnya.
  - Jenis looping pada C++ : FOR, WHILE, dan DO...WHILE.
- Penggunaan
  - Biasanya digunakan untuk menyelesaikan permasalahan yang berulang.
  - Contoh:
    - menjumlahkan angka dari 1- N
    - meminta input data berulang kepada *user*
    - Menampilkan data dengan jumlah > 1
    - dll

# Review Fungsi

- Fungsi
  - Fungsi adalah sekumpulan instruksi / pernyataan yang dikemas dalam sebuah nama yang selanjutnya dapat dipanggil di beberapa tempat dalam program.
- Tujuan
  - pembuatan fungsi dimaksudkan untuk memudahkan dalam pengembangan program.

# Review Fungsi(2)

- Keuntungan penggunaan fungsi dalam program yaitu program akan memiliki struktur yang jelas (mempunyai readability yang tinggi) dan juga akan menghindari penulisan bagian program yang sama.

# Tahapan Pembuatan Fungsi

1. Mendefinisikan fungsi
   - Memberikan nama
   - Mendefinisikan parameter formal (parameter input)
   - Mendefinisikan type hasil
2. Merealisasikan fungsi
   - Membuat algoritma fungsi: memroses input  hasil
3. Menggunakan fungsi dalam program utama
   - Memanggil fungsi dengan menggunakan parameter aktual

# Mendefinisikan Fungsi

List parameter **input** / parameter **formal**

Parameter input boleh ada, boleh kosong

```
type_hasil nama_fungsi ( [type_parameter1   nm_parameter1,
                          type_parameter2   nm_parameter2,
                          ...
                          type_parametern   nm_parametern]);

//Jelaskan spesifikasi fungsi
```

```
int fxkuadrat (int x);
//Menghasilkan x * x + 3 * x - 5
```

**Nama fungsi** : fxkuadrat
**Parameter masukan** : 1 buah, yaitu x dengan type int
**Hasil** : bertype int

# Merealisasikan Fungsi

```
type-hasil nama_fungsi ( [type-parameter1  nm-parameter1,
                          type-parameter2  nm-parameter2,
                          ...
                          type-parametern  nm-parametern] );
// Jelaskan spesifikasi fungsi
{
    // KAMUS LOKAL
    // Deklarasikan semua NAMA yang dipakai dalam algoritma
    // fungsi

    // ALGORITMA
    // Deretan teks algoritma :
    // pemberian harga, analisa kasus, pengulangan, dll.

    // Pengiriman harga di akhir fungsi, harus sesuai dengan
    // type hasil, caranya adalah:
    return (hasil);

}
```

# Kode Fungsi dalam Program

```
//Judul dan spesifikasi program
#include <iostream>
using namespace std;

DEKLARASI FUNGSI

// PROGRAM UTAMA
int main () {
        PEMAKAIAN FUNGSI
        return 0;
}

REALISASI FUNGSI
```

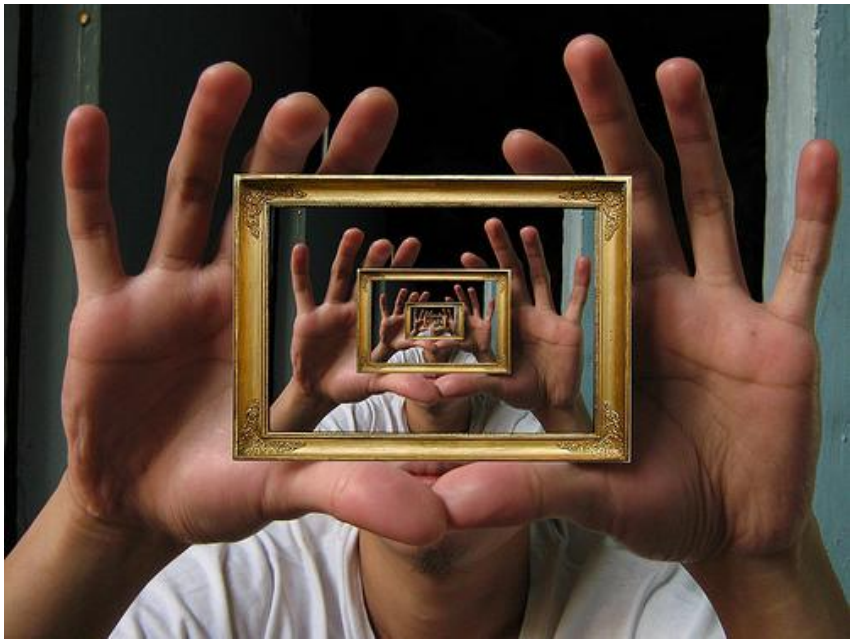
Dalam REALISASI FUNGSI bisa terdapat pemakaian fungsi lan


Q & A

# Dasar-Dasar Rekursif

# Definisi Rekursif

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



# Definisi Rekursif

- Method yang memanggil dirinya sendiri baik secara langsung maupun secara tidak langsung.
  - f(0) = 0; f(x) = 2 f(x-1) + $x^2$
    - f(1) = 1; f(2) = 6; f(3) = 21; f(4) = 58
  - Fib(0)=0; fib(1)=1; untuk n>1:fib(n) = fib(n - 1) + fib(n – 2)

# Definisi Rekursif (2)

- Fungsi yang memanggil dirinya, secara langsung atau lewat fungsi lain, disebut fungsi rekursif
- Proses pemanggilan diri itu disebut rekursi (*recursion*).
- Contoh:
  - Memangkatkan bilangan real tak nol dengan suatu pangkat bilangan bulat

$$x^n = \begin{cases} 1 & jika\ n = 0 \\ x.x^{n-1} & jika\ n > 0 \\ 1.x^{-n} & jika\ n < 0 \end{cases}$$

# Algoritma Rekursif

- Ciri masalah yang dapat diselesaikan secara rekursif adalah masalah itu dapat di-reduksi menjadi satu atau lebih masalah-masalah serupa yang lebih kecil
- Secara umum, algoritme rekursif selalu mengandung dua macam kasus:
  - **kasus induksi: satu atau lebih kasus yang pemecahan masalahnya dilakukan dengan menyelesaikan masalah serupa yang lebih sederhana (yaitu menggunakan *recursive calls*)**
  - **kasus dasar atau kasus penyetop (*base case*): satu atau lebih kasus yang sudah sederhana sehingga pemecahan masalahnya tidak perlu lagi menggunakan *recursive-calls*.**
- Supaya tidak terjadi rekursi yang tak berhingga, setiap langkah rekursif haruslah mengarah ke kasus penyetop (*base case*).

# Aturan Rekursif

1. Punya kasus dasar
   - Kasus yang sangat sederhana yang dapat memproses input tanpa perlu melakukan rekursif (memanggil method) lagi
2. Rekursif mengarah ke kasus dasar
3. "You gotta believe". Asumsikan rekursif bekerja benar.

   Pada proses pemanggilan rekursif, asumsikan bahwa pemanggilan rekursif (untuk problem yang lebih kecil) adalah benar.
   - Contoh: **pangkatRekursif (x, n)**
     - Asumsikan: **pangkatRekursif(x, n - 1)** menghasilkan nilai yang benar
     - Nilai tersebut harus diapakan sehingga menghasilkan nilai **pangkatRekursif (x, n)**yang benar?
     - Jawabannya: dikalikan dengan **x**
4. **Aturan penggabungan: Hindari duplikasi pemanggilan rekursif untuk sub-problem yang sama.**


Tanya
Jawab

# Komponen 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 memanggilnya

# Contoh Fungsi Rekursif (Faktorial)

- 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
```

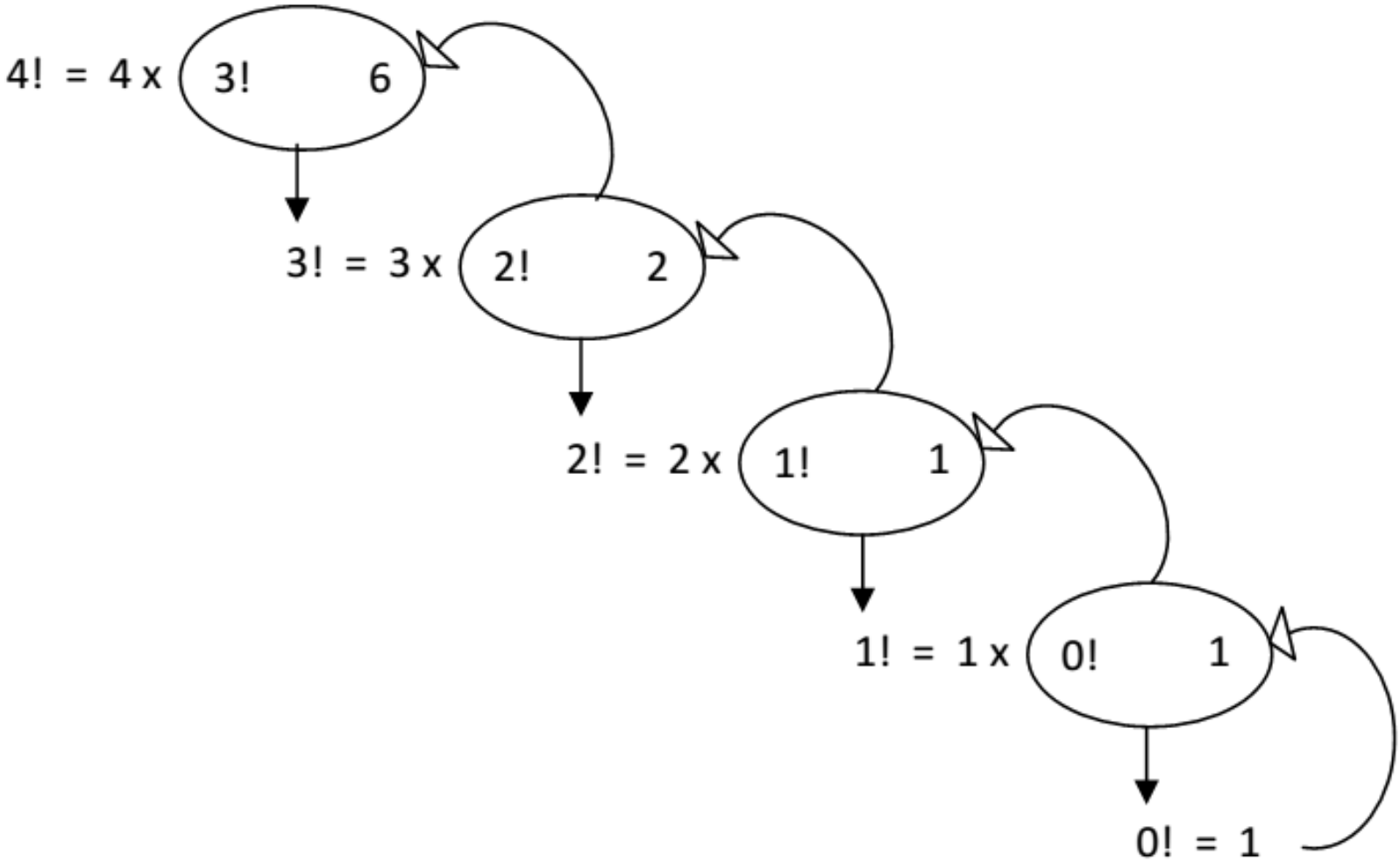
- *Faktorial - Iteratif*

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

# No. 1 dalam bhs C++ (lengkap)

```
#include<iostream.h>
int S(int n);
main()
{
  int n;
  cout << "Masukkan n = "; cin >> n;
  cout << "Deret S=1+2+3+4+5+...+n \n";
  cout << "Jumlah deret S = " << S(n);
}
int S(int n)
{
     if (n == 1)
               return (1);
     else
               return (n + S(n-1));
}
```

# Ilustrasi Rekursif



# 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 Rekursif

- 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*.

# Rekursif vs Iteratif

- Secara umum, hanya jika :
  - Penyelesaian sulit dilaksanakan secara iterative
  - 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) "


Q&A
You have
Questions
We have
Answers

# Soal Latihan

- Buatlah program untuk menyelesaikan masalah berikut:
  1. Membentuk barisan bilangan Fibonacci.
  2. Menghitung nilai kombinasi.
  3. Menghitung nilai permutasi.
  4. Menghitung nilai perpangkatan $X^n$
  5. Menghitung nilai deret angka 1+2+3+4+5+6+….
  6. Menghitung niali deret angka 2+4+6+8+10+…..
  7. Menghitung niali deret angka 1+3+5+7+9+…..
- dengan menggunakan metode rekursif.
- Penjelasan mengenai algoritma permasalahan tersebut terdapat pada slide-slide berikutnya

# No.1 Bilangan Fibonacci

- Fungsi lain yang dapat diubah ke bentuk rekursif adalah perhitungan Fibonacci. Bilangan Fibonacci dapat didefinisikan sebagai berikut:

$$f_n = f_{n-1} + f_{n-2} \text{ untuk } n > 2$$

$$f_1 = 1$$

$$f_2 = 1$$

Berikut ini adalah barisan bilangan Fibonacci mulai dari n=1

1 1 2 3 5 8 13 21 34

# Algoritma Fibonacci yang dipakai

```
Function Fibonacci(input n:integer) → integer
Deklarasi Lokal
   {tidak ada}
Deskripsi
   If (n ==1 || n==2) Then
       return (1)
   Else
       return (Fibonacci(n-1)+Fibonacci(n-2))
   Endif
```

# Contoh

- Untuk ukuran n= 4, proses perhitungan Fibonacci dapat dilakukan sebagai berikut:

$$f_4 = f_3+f_2$$
$$f_4 = (f_2+f_1) + f_2$$
$$f_4 = (1+1) +1$$
$$f_4 = 3$$

# No.2 Kombinasi

```
Function Kombinasi (input n, r : integer) → real
Deklarasi
 If (n < r) Then
  return (0)
 Else
  return (Faktorial(n)/Faktorial(r)*Faktorial(n-r))
 Endif
```

# No.3 Permutasi

```
Function Permutasi (input n, r : integer) → real
Deklarasi
    {tidak ada}
 Deskripsi
    If (n< r) Then
          return (0)
    Else
          return (Faktorial(n) / Faktorial(n-r))
    Endif
```

# No.4 deret S=1+2+3+4+5+...+n

```
Function S(input n:integer) → integer
Deklarasi Lokal
   {tidak ada}
Deskripsi
   If (n==1) Then
      return (1)
   Else
      return (n + S(n-1))
   Endif
```

# No.5 deret S=2+4+6+8+10+...+2n

```
Function S(input n:integer) → integer
Deklarasi Lokal
   {tidak ada}
Deskripsi
   If (n==1) Then
      return (2)
   Else
      return (2*n + S(n-1))
   Endif
```

# No.6 deret S=1+3+5+7+9+...+2n-1

```
Function S(input n:integer) → integer
Deklarasi Lokal
   {tidak ada}
Deskripsi
   If (n==1) Then
      return (1)
   Else
      return ((2*n)-1 + S(n-1))
   Endif
```

# Pre-test

# Latihan

# Rangkuman Materi

# Tugas