

# Skema Standar (Bag. 2): Skema Pengulangan dan Pemrosesan Sekuensial

Tim Pengajar IF1210

Sekolah Teknik Elektro dan Informatika

# SKEMA PENGULANGAN

# Jenis-Jenis Skema Pengulangan

1. Berdasarkan jumlah pengulangan
2. Berdasarkan kondisi berhenti
3. Berdasarkan kondisi mengulang
4. Berdasarkan dua aksi
5. Berdasarkan pencacah

# Contoh Persoalan

- Tuliskan sebuah program yang membaca sebuah nilai  $N$  (integer positif,  $> 0$ ) dan menuliskan output nilai 1, 2, 3, ...,  $N$  berderet ke bawah sbb.

1

2

3

...

$N$

- Contoh:  $N = 3$ , outputnya adalah:

1

2

3

- Contoh-2:  $N = 1$ , outputnya adalah:

1

# Skema Pengulangan – 1

## Berdasarkan jumlah pengulangan

- Notasi Algoritmik:

<p><u><b>repeat</b></u> <b>n</b> <u><b>times</b></u> <i>&lt;aksi&gt;</i></p>
--

- Penggunaan:
  - Diketahui secara persis **berapa kali** aksi harus dilakukan dan aksi terdefinisi

# Skema Pengulangan – 1: Contoh

```
Program TulisBill  
{ Dibaca N>0, dituliskan 1,2,3,...,N berderet ke bawah }  
KAMUS  
    N, i : integer  
ALGORITMA  
    input (N)  
    i ← 1  
    repeat N times  
        output (i)  
        i ← i + 1
```

- Bentuk pengulangan *repeat N times* kurang sesuai
  - Tidak benar-benar melakukan aksi mencetak sesuatu N kali, dibutuhkan variabel i untuk *increment* nilai yang dicetak

# Skema Pengulangan – 2

## Berdasarkan kondisi berhenti

- Notasi Algoritmik:

```
repeat  
    <aksi>  
until <kondisi-berhenti>
```

- Penggunaan:
  - Aksi minimum dilakukan 1x (karena kondisi berhenti baru dicek setelah aksi dieksekusi)
  - Kondisi berhenti berupa ekspresi boolean (lebih luas dari sekedar harus mengulang berapa kali)

# Skema Pengulangan – 2: Contoh

**Program** TulisBil2

{ Dibaca  $N > 0$ , dituliskan 1,2,3,...,N berderet ke bawah }

**KAMUS**

N, i : integer

**ALGORITMA**

input (N)

i  $\leftarrow$  1

repeat

output (i)

    i  $\leftarrow$  i + 1

until (i > N)

- Harus dijamin nilai N yang dibaca sesuai spesifikasi yaitu  $N > 0$ 
  - Jika nilai N tidak diberikan sesuai spesifikasi, akan tercetak angka 1 di layar
- Nilai i yang terdefinisi: 1..N+1
- Alternatif: mendefinisikan i sebagai bilangan yang **sudah** ditulis dan memulai i dengan 0 sehingga nilai i = 0..N



# Skema Pengulangan – 3

## Berdasarkan kondisi mengulang

- Notasi Algoritmik:

```
while <kondisi-mengulang> do  
    <aksi>  
{ Kondisi berhenti dicapai di sini }
```

- Penggunaan:
  - Dimungkinkan aksi tidak pernah dilakukan (karena kondisi mengulang dicek sebelum aksi dilakukan) → kasus yang menyebabkan aksi tidak pernah dilakukan disebut sbg. **kasus kosong**

# Skema Pengulangan – 3: Contoh

**Program** TulisBil3

{ Dibaca  $N > 0$ , dituliskan 1, 2, 3, ..., N berderet ke bawah }

**KAMUS**

N, i : integer

**ALGORITMA**

input(N)

i  $\leftarrow$  1

while (i  $\leq$  N) do

output(i)

    i  $\leftarrow$  i + 1

{ i > N }

- Jika pengguna memberikan nilai N negatif atau 0, program tidak menuliskan apa pun
  - Kondisi diperiksa sebelum memasuki loop, yaitu  $i \leq N$  bernilai false
- Nilai i didefinisikan 1..N+1
- Alternatif: mendefinisikan i sebagai bilangan yang **sudah** ditulis dan memulai i dengan 0 sehingga nilai i = 0..N

# Skema Pengulangan – 4

## Berdasarkan dua aksi (1)

- Notasi Algoritmik:

**iterate**

*<aksi-1>*

**stop** *<kondisi-berhenti>*

*<aksi-2>*

{ Kondisi berhenti dicapai di sini }

- Penggunaan:
  - Merupakan gabungan antara repeat-until dan while-do
  - aksi-1 minimum dilakukan 1x dan aksi-2 bisa tidak dilakukan sama sekali

# Skema Pengulangan – 4: Contoh

**Program** TulisBil4

{ Dibaca  $N > 0$ , dituliskan 1,2,3,...,N  
berderet ke bawah }

**KAMUS**

N, i : integer

**ALGORITMA**

input (N)

i  $\leftarrow$  1

iterate

output (i)

stop (i = N)

i  $\leftarrow$  i + 1

{ i = N }

- Harus dijamin nilai N yang dibaca sesuai spesifikasi yaitu  $N > 0$ 
  - Jika nilai N tidak diberikan sesuai spesifikasi, akan tercetak angka 1 di layar
- Nilai i didefinisikan 1..N

# Skema Pengulangan – 5

## Berdasarkan pencacah

- Notasi algoritmik:

```
{ Mencacah maju }  
<pencacah> traversal [<min>..<maks>]  
  <aksi>  
{ Mencacah mundur }  
<pencacah> traversal [<maks>..<min>]  
  <aksi>
```

- Penggunaan

- Pencacah harus bertipe ordinal, contoh: integer
- Diketahui dengan pasti nilai awal dan nilai akhir pencacah

# Skema Pengulangan – 5: Contoh

```

Program TulisBil5
{ Dibaca N>0, dituliskan 1,2,3,...,N
berderet ke bawah }
KAMUS
    N, i : integer
ALGORITMA
    input(N)
    i traversal [1..N]
        output(i)
  
```

- Bentuk yang ekuivalen dengan iterate-stop untuk  $i = 1..N$ , namun pertambahan  $i$  ditangani secara implisit oleh pemroses bahasa
- Jika nilai  $N$  yang diberikan oleh pengguna adalah 0 atau negatif, maka akan terjadi penulisan  $i$  yang di luar definisi karena terjadi pencacahan mundur
  - Contoh: jika  $N = 0$ , maka akan tercetak:

1  
0

# Catatan

- Ada bermacam-macam pengulangan
  - satu bentuk pengulangan dapat "diterjemahkan" menjadi bentuk yang lain dengan notasi algoritmik yang tersedia.
- Tidak semua bahasa pemrograman yang ada menyediakan semua bentuk pengulangan
  - Contoh: Python sebenarnya hanya punya 2 bentuk dasar pengulangan: for, while
- Instruksi pengulangan tidak dapat berdiri sendiri
  - Harus disertai dengan instruksi-instruksi lain sebelum dan sesudah pengulangan.
- Persoalannya adalah: **memilih bentuk pengulangan yang benar dan tepat untuk kelas persoalan tertentu**
  - Inti dari desain algoritmik

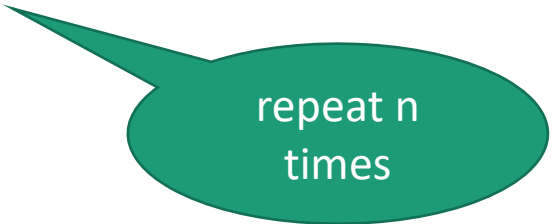
# Latihan 1. Skema Pengulangan (1)

1. Mengisikan elemen sebuah array of integer hingga penuh, diketahui indeks awal 1 dan indeks akhir array 100.



traversal

2. Menuliskan ke layar sebuah kata “Hello” sebanyak  $n$  kali,  $n$  masukan dari pengguna.



repeat  $n$   
times



# Latihan 1. Skema Pengulangan (2)

3. Memunculkan sebuah menu ke layar, yaitu:

- Menampilkan “Hello World” di layar,
- Membaca 2 integer dan menampilkan hasil penjumlahannya
- Keluar

Program akan selesai hanya jika pengguna memilih menu “Keluar”. Jika memilih menu yang lain, program akan mengeksekusi perintah menu tersebut, lalu kembali ke tampilan menu, untuk selanjutnya diulangi lagi proses yang sama.



iterate-stop

# Latihan 1. Skema Pengulangan (3)

4. Mengisi sebuah array of integer dengan ukuran N. Pengisian array diakhiri jika pengguna memasukkan angka 9999.
- Array mungkin menjadi kosong (jika angka yang dimasukkan pertama kali 9999).
  - Tidak ada penanganan khusus untuk kasus array kosong

while-do

# Translasi Skema Pengulangan dalam Bahasa C

# Pengulangan

## Notasi Algoritmik

**Pengulangan berdasarkan kondisi berhenti:**

```
repeat
    Aksi
until kondisi-stop
```

**Pengulangan berdasarkan kondisi ulang:**

```
while (kondisi-ulang) do
    Aksi
{not kondisi-ulang}
```

## Bahasa C

```
do {
    Aksi;
} while (!kondisi-stop);
```

```
while (kondisi-ulang) {
    Aksi;
} /*not kondisi-ulang */
```

# Pengulangan

## Notasi Algoritmik

**Pengulangan berdasarkan pencacah:**

i traversal [Awal..Akhir]  
Aksi

## Bahasa C

```
/* Jika Awal <= Akhir */
for(i=Awal;i<=Akhir;i++) {
    Aksi;
}
/* Jika Awal >= Akhir */
for(i=Awal;i>=Akhir;i--) {
    Aksi;
}
```

**Catatan:**

```
for(exp1;exp2;exp3) {
    Aksi;
}
ekivalen dengan:
exp1;
while (exp2) {
    Aksi;
    exp3;
} /* !exp2 */
```

# Pengulangan

## Notasi Algoritmik

**Pengulangan berdasarkan dua aksi:**

iterate

Aksi-1

stop kondisi-stop

Aksi-2

## Bahasa C

```
for(;;) {  
    Aksi-1;  
    if (kondisi-stop)  
        break;  
    else  
        Aksi-2;  
}
```

# SKEMA PEMROSESAN SEKUENSIAL

# Skema Pemrosesan Sekuensial (1)

- **Pemrosesan sekuensial** adalah pemrosesan secara satu persatu, dari sekumpulan informasi sejenis yang setiap elemennya dapat diakses dengan keterurutan tertentu (ada suksesor)
- Jadi seakan-akan **kumpulan elemen merupakan “deret” elemen**
- Type elemen yang akan diproses:
  - type dasar
  - type bentukan



## Skema Pemrosesan Sekuensial (2)

- Kumpulan informasi disimpan sedemikian rupa, sehingga selalu dikenali:
  - Elemen pertama (**First\_Elmt**)
  - Elemen yang siap diproses (**Current\_Elmt**)
  - Elemen yang diakses setelah Current\_Elmt (**Next\_Elmt**)
  - Tanda akhir proses (**EOP**)
- Bagaimana EOP bernilai true?
  - Model **dengan MARK**: elemen terakhir adalah elemen “fiktif”, sebetulnya bukan anggota elemen yang diproses
  - Model **tanpa MARK**: elemen terakhir mengandung info yang memberitahukan bahwa elemen tsb. adalah elemen terakhir

# Skema Pemrosesan Sekuensial Dengan MARK (1)

## **SKEMA PEMROSESAN SEKUENSIAL DENGAN MARK**

{ Tanpa penanganan kasus kosong secara khusus }

### **SKEMA**

*Inisialisasi*

*First\_Elmt*

while not (EOP) do

*Proses\_Current\_Elmt*

*Next\_Elmt*

{ EOP }

*Terminasi*

# Skema Pemrosesan Sekuensial Dengan MARK (2)

## **SKEMA PEMROSESAN SEKUENSIAL DENGAN MARK**

{ Dengan penanganan kasus kosong }

### **SKEMA**

```
First_Elmt  
if (EOP) then  
    Proses_Kasus_Kosong  
else  
    Inisialisasi  
    repeat  
        Proses_Current_Elmt  
        Next_Elmt  
    until (EOP)  
    Terminasi
```

# Skema Pemrosesan Sekuensial Tanpa MARK (1)

## **SKEMA PEMROSESAN SEKUENSIAL TANPA MARK**

{ Karena tanpa mark, tak ada kasus kosong. }  
{ Dengan **iterate-stop** }

### **SKEMA**

*Inisialisasi*

*First\_Elmt*

iterate

*Proses\_Current\_Elmt*

stop (EOP)

*Next\_Elmt*

*Terminasi*

# Skema Pemrosesan Sekuensial Tanpa MARK (2)

## **SKEMA PEMROSESAN SEKUENSIAL TANPA MARK**

```
{ Karena tanpa mark, tak ada kasus kosong.  
  Proses elemen pertama tidak berbeda dengan akses Next_Elmt }
```

## **SKEMA**

*Inisialisasi*

repeat

*Next\_Elmt*

*Proses\_Current\_Elmt*

until (*EOP*)

*Terminasi*

# Studi Kasus 1: Jumlah 1 s.d. N

- Buatlah algoritma yang membaca sebuah bilangan bulat positif N, menuliskan: 1, 2, 3, ..., N dan menjumlahkan  $1 + 2 + 3 + \dots + N$  serta menuliskan hasil penjumlahan.
- Berikut beberapa versi program dan penjelasannya.

# Studi Kasus 1 – Versi-2

## Model tanpa Mark

### **Program** SUMNBil1

{ Menjumlahkan  $1+2+3+...+N$  dengan N dibaca.

**Model tanpa mark }**

### **KAMUS**

i : integer            { bilangan yang akan dijumlahkan }  
 N : integer > 0    { banyaknya bilangan yang dijumlahkan }  
 Sum : integer        { jumlah }

### **ALGORITMA**

input(N); Sum  $\leftarrow$  0            { Inisialisasi }  
 i  $\leftarrow$  1                            { First Element }  
iterate  
     output(i)                        { Proses current element }  
     Sum  $\leftarrow$  Sum + i            { Proses current element }  
stop (i = N)                        { EOP: i = N }  
     i  $\leftarrow$  i + 1                    { Next Elmt }  
output(Sum)                        { Terminasi }

# Studi Kasus 1 – Versi-1: Diskusi

- Pengontrol pengulangan adalah bilangan integer  $i$
- Analisis:
  - Model tanpa MARK
  - EOP adalah jika  $i = N$
  - First\_Elmt = 1
  - Next\_Elmt =  $i+1$
- Program **benar dengan tambahan spesifikasi  $N \geq 1$**  sesuai spesifikasi domain  $i = 1..N$ .
  - Nilai  $i$  menaati domain  $1..N$



# Studi Kasus 1 – Versi-2

## Model dengan Mark, tanpa penanganan kasus kosong

### **Program** SUMNBil2

{ Menjumlahkan  $1+2+3+\dots+N$  dengan N dibaca.

**Model dengan mark, tanpa penanganan kasus kosong }**

### **KAMUS**

i : integer            { bilangan yang akan dijumlahkan }  
 N : integer > 0    { banyaknya bilangan yang dijumlahkan }  
 Sum : integer        { jumlah }

### **ALGORITMA**

```
input(N); Sum ← 0            { Inisialisasi }
i ← 1                        { First Element }
while (i ≤ N) do            { EOP: i > N }
    output(i)                 { Proses current element }
    Sum ← Sum + i            { Proses current element }
    i ← i + 1                { Next Elmt }
{ i > N, i = N + 1, Sum = 1 + 2 + 3 + ... + N }
output(Sum)                 { Terminasi }
```

# Studi Kasus 1 – Versi-2: Diskusi

- Pengontrol pengulangan adalah bilangan integer  $i$
- Analisis:
  - Model dengan MARK
  - EOP adalah jika  $i > N$ , yaitu jika  $i = N+1$
  - $\text{First\_Elmt} = 1$
  - $\text{Next\_Elmt} = i+1$
- Program **benar tanpa batasan nilai  $N$** . Jika  $N \leq 0$ , terjadi kasus kosong.
- Nilai  $i$  berada di luar definisi domain  $1..N$ , yaitu  $1..N+1$

# Studi Kasus 1 – Versi-3

## Model tanpa Mark

### **Program** SUMNBil3

{ Menjumlahkan  $1+2+3+\dots+N$  dengan N dibaca.

**Model tanpa mark }**

### **KAMUS**

i : integer            { bilangan yang akan dijumlahkan }  
 N : integer > 0    { banyaknya bilangan yang dijumlahkan }  
 Sum : integer        { jumlah }

### **ALGORITMA**

input(N); Sum  $\leftarrow$  0            { Inisialisasi }  
 i  $\leftarrow$  1                        { First Element }  
repeat  
     output(i)  
     Sum  $\leftarrow$  Sum + i  
     i  $\leftarrow$  i + 1                { Next Elmt }  
until (i > N)  
 { i > N, i = N + 1, Sum = 1 + 2 + 3 + ... + N }  
output(Sum)                      { Terminasi }

# Studi Kasus 1 – Versi-3: Diskusi

- Pengontrol pengulangan adalah bilangan integer  $i$
- Analisis:
  - Model tanpa MARK
  - EOP adalah jika  $i > N$ , yaitu jika  $i = N+1$
  - First\_Elmt = 1
  - Next\_Elmt =  $i+1$
- Program **benar dengan tambahan spesifikasi  $N \geq 1$**
- Nilai  $i$  di luar definisi domain  $1..N$ , yaitu  $1..N+1$

# Studi Kasus 1 – Versi-4

## Model tanpa Mark

### **Program** SUMNBil3

{ Menjumlahkan  $1+2+3+\dots+N$  dengan N dibaca.

**Model tanpa mark }**

### **KAMUS**

i : integer { bilangan yang akan dijumlahkan }

N : integer > 0 { banyaknya bilangan yang dijumlahkan }

Sum : integer { jumlah }

### **ALGORITMA**

input(N); Sum  $\leftarrow$  0 { Inisialisasi }

i traversal [1..N]

output(i)

Sum  $\leftarrow$  Sum + i

{ i = ?, Sum =  $1 + 2 + 3 + \dots + N$  }

output(Sum) { Terminasi }

# Studi Kasus 1 – Versi-4: Diskusi

- Pengontrol pengulangan adalah bilangan integer  $i$
- Analisis:
  - Model tanpa MARK
  - EOP adalah jika  $i = N$
  - $\text{First\_Elmt} = 1$
  - $\text{Next\_Elmt} = i+1$
- Program **benar dengan tambahan spesifikasi  $N \geq 1$**
- Nilai  $i$  sesuai definisi domain  $1..N$ , namun setelah traversal nilai  $i$  tidak terdefinisi, nilai  $i$  tidak boleh digunakan setelah traversal berakhir

## Studi Kasus 2: Penjumlahan Deret Bilangan

- Tuliskan sebuah program yang membaca nilai-nilai integer yang dibaca dari piranti masukan dan menjumlahkan nilainya. Pemasukan nilai integer diakhiri dengan 9999.
- Berikut beberapa versi program dan penjelasannya.

# Versi-1

## Model dengan mark, tanpa penanganan kasus kosong

### **Program** JumBilX1

{ Menjumlahkan nilai-nilai X yang dibaca. Mark = 9999.

**Model dengan mark, tanpa penanganan kasus kosong. }**

### **KAMUS**

X : integer            { sekumpulan bilangan integer yang dibaca untuk  
                                 dijumlahkan, pembacaan diakhiri dengan 9999 }

Sum : integer        { jumlah }

### **ALGORITMA**

Sum  $\leftarrow$  0                            { Inisialisasi }

input(X)                                { First Element }

while (X  $\neq$  9999) do                { EOP: X = 9999 }

output(X)

    Sum  $\leftarrow$  Sum + X                { Proses current element }

input(X)                            { Next Elmt }

{ Sum =  $X_1 + X_2 + X_3 + \dots + X_N$  }

output(Sum)                            { Terminasi }



## Studi Kasus 2 – Versi-1: Diskusi

- Pengontrol pengulangan (*current element*) adalah nilai X
- Program benar untuk kasus kosong maupun tidak kosong.
  - Jika nilai yang diketikkan langsung 9999 (kasus kosong), tidak ada yang diproses → namun demikian, **tidak ada penanganan terhadap kasus kosong.**
  - Program tidak salah jika tidak terjadi kasus kosong: minimal 1 nilai X dibaca dan diproses
- Skema yang dipilih cukup baik
  - Dengan tambahan spesifikasi: Jika terjadi kasus kosong, jumlah bilangan = 0

# Model dengan mark, dengan penanganan kasus kosong

{ Menjumlahkan nilai-nilai X yang dibaca. Mark = 9999.

Model dengan mark, dengan penanganan kasus kosong. }

## KAMUS

```
X : integer      { sekumpulan bilangan integer yang dibaca untuk
                    dijumlahkan, pembacaan diakhiri dengan 9999 }
```

```
Sum : integer { jumlah }
```

## ALGORITMA

```
input(X)           { First Element }
```

```
if (X = 9999) then
```

```
output("Kasus kosong: yang diketik langsung 9999")
```

```
else { X ≠ 9999 }
```

```
Sum ← 0           { Inisialisasi }
```

repeat

output (X)

$$\text{Sum} \leftarrow \text{Sum} + X$$

```
input(X)           { Next Elmt }
```

```
until (X = 9999)
```

$$\{ \text{Sum} = X_1 + X_2 + X_3 + \dots + X_N \}$$

```
output (Sum)           { Terminasi }
```



## Studi Kasus 2 – Versi-2: Diskusi

- Pengontrol pengulangan (*current element*) adalah nilai X
- Program benar untuk kasus kosong maupun tidak kosong.
  - Jika nilai yang diketikkan langsung 9999 (kasus kosong), diberikan pesan kesalahan (penanganan terhadap kasus kosong)
  - Jika bukan, minimal satu nilai dibaca dan diproses.
- Skema yang dipilih **tepat**

# Studi Kasus 3: Jumlahkan dan Cacah Bilangan

- Tuliskan sebuah program yang membaca sekumpulan nilai integer yang diketikkan lewat piranti masukan dan sekaligus melakukan:
  - Penjumlahan dari nilai integer yang dibaca,
  - Mencacah (melakukan *counting*) banyaknya nilai integer yang dibaca
- Pemasukan nilai integer diakhiri dengan 9999.
- Pada akhir program, harus dituliskan jumlah dan banyaknya integer yang dibaca
- Berikut adalah beberapa versi program dan penjelasannya.



## Studi Kasus 3 – Versi-1: Diskusi

- Pengontrol pengulangan (*current element*) adalah nilai X
- Program **memakai skema yang benar**, namun tidak menangani kasus kosong
  - Jika nilai yang diketikkan langsung 9999 (kasus kosong), tidak terjadi apa pun  
→ tidak ada penanganan kasus kosong
  - Jika bukan, minimal satu nilai dibaca dan diproses.
- Program benar dengan tambahan spesifikasi:
  - Banyaknya bilangan = 0 dan jumlah bilangan = 0, jika terjadi kasus kosong

# Versi-2

## Model tanpa mark dengan repeat-until

### Program SumXBil2

{ Menjumlahkan dan mencacah nilai-nilai X yang dibaca, Mark = 9999. }

### **KAMUS**

i : integer { banyaknya integer yang **sudah** dibaca }

X : integer { sekumpulan bilangan integer yang dibaca, diakhiri dengan 9999 }

Sum : integer { jumlah }

### **ALGORITMA**

i ← 0; Sum ← 0 { Inisialisasi }

input(X) { First Element }

repeat

output(X)

Sum ← Sum + X

i ← i + 1

input(X) { Next Elmt }

until (X = 9999) do

{ i = bilangan ke... yang **sudah** dibaca, Sum =  $X_1 + X_2 + X_3 + \dots + X_i$  }

output("Jumlah : ", Sum) { Terminasi }

output("Banyaknya bilangan : ", i)

## Studi Kasus 3 – Versi-2: Diskusi

- Pengontrol pengulangan (*current element*) adalah nilai X
- Program **memakai skema yang salah**
  - Jika nilai yang diketikkan langsung 9999 (kasus kosong), mark akan diproses sehingga program menjadi salah
  - Program tidak salah jika tidak terjadi kasus kosong (nilai yang diketikkan minimal diproses satu kali)



# Versi-3: Model dengan mark, dengan penanganan kasus kosong

## **Program** SumXBil3

{ Menjumlahkan dan mencacah nilai-nilai X yang dibaca, Mark = 9999. }

## **KAMUS**

i : integer { banyaknya integer yang **sudah** dibaca }

X : integer { sekumpulan bilangan integer yang dibaca, diakhiri dengan 9999 }

Sum : integer { jumlah }

## **ALGORITMA**

input(X) { First Element }

if (X = 9999) then

output("Kasus kosong: yang diketik langsung 9999")

else { X ≠ 9999 }

    i ← 0; Sum ← 0 { Inisialisasi }

repeat

output(X)

        Sum ← Sum + X

        i ← i + 1

input(X) { Next Elmt }

until (X = 9999)

    { i = bilangan ke... yang **sudah** dibaca, Sum =  $X_1 + X_2 + X_3 + \dots + X_i$  }

output("Jumlah : ", Sum) { Terminasi }

output("Banyaknya bilangan : ", i)

## Studi Kasus 3 – Versi-3: Diskusi

- Pengontrol pengulangan (*current element*) adalah nilai X
- Program **memakai skema yang benar**, dengan penanganan kasus kosong
  - Jika nilai yang diketikkan langsung 9999 (kasus kosong), diberikan pesan kesalahan (penanganan terhadap kasus kosong)
  - Jika bukan, minimal satu nilai dibaca dan diproses.
- Semua invarian terpenuhi. Definisi dan inisialisasi i tepat.

# Kesimpulan

- Skema pemrosesan sekuensial **dengan mark** → ada pemeriksaan terhadap **mark** terlebih dahulu sebelum pemrosesan lebih lanjut
  - Memungkinkan kasus kosong
- Skema pemrosesan sekuensial **tanpa mark** → setiap elemen diproses minimum 1 kali
  - Tidak ada kasus kosong
- Pada prinsipnya skema pengulangan dan pemrosesan sekuensial apa pun dapat dipakai, tapi ingat *nature* dari masing-masing skema

## Latihan 2. Skema Pemrosesan Sekuensial (1)

- Buatlah program dalam notasi algoritmik, untuk membaca sejumlah nilai UTS mahasiswa di suatu kelas. Nilai UTS yang valid adalah 0..100. Pembacaan dihentikan jika **masukan nilai UTS di luar range nilai yang diizinkan**
- Di akhir program dihitung dan ditampilkan rata-rata nilai UTS seluruh mahasiswa di kelas.
- Jika tidak ada nilai UTS yang dimasukkan, tuliskan “Tidak ada data”

## Latihan 2. Skema Pemrosesan Sekuensial (2)

Contoh masukan dan keluaran:

Catatan: yang bergaris bawah adalah masukan pengguna

### Contoh 1

Nilai UTS = 50

Nilai UTS = 100

Nilai UTS = 9999

Nilai rata-rata UTS = 75

### Contoh 2

Nilai UTS = 101

Tidak ada data

Diskusi:  
Skema apa yang  
sebaiknya digunakan

# SKEMA PROSES VALIDASI II

# Skema Validasi II (1)

## **SKEMA VALIDASI MASUKAN**

```
{ Data masukan divalidasi shg didapatkan data yang valid }  
{ Proses tanpa mark, dengan iterate-stop }  
{ Jika data tidak valid, diberikan pesan kesalahan }
```

## **SKEMA**

```
iterate  
    input(<data>)  
stop(<data_valid>)  
    <pesan_kesalahan>  
  
<proses_data_valid>
```

## Skema Validasi II (2)

### SKEMA VALIDASI MASUKAN

```
{ Data masukan divalidasi shg didapatkan data yang valid }  
{ Proses tanpa mark, dengan repeat-until }  
{ Jika data tidak valid, tidak ada pesan kesalahan }
```

### SKEMA

```
repeat  
    input (<data>)  
until (<data_valid>)  
  
<proses_data_valid>
```



## Contoh Persoalan: Ranging-2

- Buatlah program yang membaca 3 buah integer a, b, dan c dan menuliskan secara terurut mulai dari terkecil s.d. yang terbesar.
- Ketiga bilangan yang dibaca harus berlainan harganya. Jika tidak, maka pembacaan input harus diulangi sampai didapatkan nilai-nilai yang berbeda.

# Contoh: Ranking-2 – Versi-1

**Program RANGKING1**

```
{ Dibaca 3 integer a, b, c. Harus divalidasi hingga didapatkan nilai yang benar yaitu a, b, c yang berbeda }  
{ Dituliskan dari terkecil s.d. terbesar }
```

**KAMUS**

a, b, c : integer

**ALGORITMA**

```
{ Validasi input }  
iterate  
    input(a,b,c)  
stop (a ≠ b and a ≠ c and b ≠ c)  
    output("Data salah, tidak sesuai spesifikasi. Ulangi.")  
{ Proses data valid }  
depend on (a,b,c)  
    a < b < c : output(a,b,c)  
    a < c < b : output(a,c,b)  
    b < a < c : output(b,a,c)  
    b < c < a : output(b,c,a)  
    c < a < b : output(c,a,b)  
    c < b < a : output(c,b,a)
```

# Contoh: Ranking-2 – Versi-2

**Program RANGKING1**

```
{ Dibaca 3 integer a, b, c. Harus divalidasi hingga didapatkan nilai yang benar yaitu a, b, c yang berbeda }  
{ Dituliskan dari terkecil s.d. terbesar }
```

**KAMUS**

a, b, c : integer

**ALGORITMA**

```
{ Validasi input }  
repeat  
    input(a,b,c)  
until (a ≠ b and a ≠ c and b ≠ c)  
{ Proses data valid }  
depend on (a,b,c)  
    a < b < c : output(a,b,c)  
    a < c < b : output(a,c,b)  
    b < a < c : output(b,a,c)  
    b < c < a : output(b,c,a)  
    c < a < b : output(c,a,b)  
    c < b < a : output(c,b,a)
```

# Latihan 3.

## Skema Proses Validasi (2)

- Buatlah program dalam notasi algoritmik yang menerima 3 buah bilangan integer yaitu h, m, dan s yang akan digunakan untuk membentuk data bertipe jam. Definisi type jam adalah sbb.

```
type jam : < HH : integer[0..23]; { bagian jam }  
           MM : integer[0..59]; { bagian menit }  
           SS : integer[0..59] > { bagian detik }
```

- **Jika ketiga input tidak valid**, dituliskan pesan kesalahan ke layar “Tidak dapat membentuk jam” dan pemasukan data h, m, s diulangi sampai didapatkan nilai yang valid.
- **Jika ketiga input valid**, maka sebuah variabel J bertipe jam akan terbentuk (didefinisikan nilainya) dengan J.HH bernilai h, J.MM bernilai m, J.SS bernilai s.
- Nilai valid didefinisikan sebagai:  $0 \leq h \leq 23$ ;  $0 \leq m \leq 59$ ;  $0 \leq s \leq 59$

# Latihan Soal

## Latihan 4 (1)

- Buatlah program dalam notasi algoritmik, untuk membaca nilai UTS dan nilai UAS mahasiswa untuk setiap pelajaran yang diikutinya (0..100) **dan diakhiri jika nilai masukan UTS di luar range nilai yang diizinkan**, kemudian menghitung dan mencetak rata-rata nilai akhir dari seluruh pelajaran.
- Gunakan skema validasi data untuk memastikan nilai UAS pada range 0..100 (jika data tidak memenuhi syarat, read data UAS diulang). Nilai akhir untuk suatu pelajaran dihitung dari rumus  $(40\% * \text{nilai UTS}) + (60\% * \text{nilai UAS})$ .
- Contoh masukan dan keluaran:

# Latihan 4 (2)

## Contoh masukan dan keluaran

Catatan: yang bergaris bawah adalah input pengguna

### Contoh 1

```
Nilai UTS = 50  
Nilai UAS = 200  
Ulangi input nilai (0..100)!  
Nilai UAS = 100  
Nilai akhir pelajaran 1 = 80  
Nilai UTS = 100  
Nilai UAS = 50  
Nilai Akhir pelajaran 2 = 70  
Nilai UTS = 9999  
Nilai rata-rata dari 2 pelajaran  
adalah = 75
```

### Contoh 2

```
Nilai UTS = 101  
Data kosong, tidak ada nilai rata-  
rata!
```

# Sumber Utama

Diktat “Dasar Pemrograman, Bag.  
Pemrograman Prosedural” oleh Inggriani  
Liem



# Selamat Belajar