



Skema Standar (Bag. 1)

Tim Pengajar

IF1210 Dasar Pemrograman

Sem. 2 2019/2020





Sebelumnya...

- Review dekomposisi data dan fungsional dalam pemrograman prosedural
- Dekomposisi fungsional:
 - Function
 - Procedure
- Dekomposisi data: menggunakan type bentukan/type komposisi/record
 - Type bentukan di Python diimplementasikan dalam bentuk tuple



SKEMA STANDAR



Apa itu skema standar?

- Skema standar adalah skema umum yang sering digunakan untuk memecahkan kasus-kasus trivial (contoh: pemasukan nilai, penampilan nilai, pencarian nilai, pengurutan, dll)
- Skema standar yang dipelajari di IF1210:
 - Skema proses validasi (I dan II)
 - Skema pengulangan
 - Skema pemrosesan sekuensial: skema tanpa mark, skema dengan mark
 - Skema pemrosesan array: skema traversal, skema pencarian nilai ekstrim, skema searching, skema sorting
 - Skema pemrosesan file I/O



Apa pentingnya skema standar?

- Penggunaan pola yang seragam antar programmer
- Mempermudah pembacaan dan pemeriksaan kode sumber oleh orang lain
- Mempercepat penulisan program untuk kasus-kasus yang umum (pengisian, penulisan, validasi, pencarian nilai ekstrim, dll)



SKEMA PROSES VALIDASI I

Skema Proses Validasi



SKEMA PROSES_VALIDASI

{Skema program yang menerima input data, hanya melakukan proses jika data valid}

KAMUS

{deklarasi data input}

ALGORITMA

```
input(data)
if (<data_valid>) then
    { <data_valid> adalah predikat, suatu ekspresi
      boolean yang menyatakan validitas dari data }
    { Lakukan_proses }
else { data tidak valid }
    output("Data salah, tidak sesuai spesifikasi")
```



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



Skema Pengulangan – 1

Berdasarkan jumlah pengulangan

- Notasi Algoritmik:

```
repeat n times  
  <aksi>
```

- Notasi Python:

```
for i in range (n) :  
  <aksi>
```

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

Skema Pengulangan – 2

Berdasarkan kondisi berhenti



- Notasi Algoritmik:

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

- Notasi Python:
 - Lihat catatan di slide berikutnya
- 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

Berdasarkan kondisi berhenti

- Notasi Python:
 - Python tidak memiliki notasi khusus untuk mewakili repeat-until
 - Adaptasi:

```
while True:  
    <aksi>  
    if (<kondisi-berhenti>) :  
        break
```



Skema Pengulangan – 3

Berdasarkan kondisi mengulang

- Notasi Algoritmik:

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

- Notasi Python:

```
while (<kondisi-mengulang>) :  
    <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 – 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

Berdasarkan dua aksi (1)



- Notasi Python:
 - Python tidak memiliki notasi khusus untuk mewakili iterate-stop
 - Adaptasi:

```
while True:
    <aksi-1>
    if (<kondisi-berhenti>) :
        break
    else:
        <aksi-2>
# Kondisi berhenti dicapai di sini
```

Skema Pengulangan – 5

Berdasarkan pencacah (1)



- Notasi Algoritmik:

$\langle \text{nama-pencacah} \rangle$ <u>traversal</u> [$\langle \text{range-harga} \rangle$] $\langle \text{aksi} \rangle$
--

- Notasi Python:

```
# <nama-pencacah> = i

# Jika Awal <= Akhir
for i in range (Awal,Akhir+1):
    <aksi>

# Jika Awal >= Akhir (mundur)
for i in range (Akhir,Awal-1,-1):
    <aksi>
```


Skema Pengulangan – 5

Berdasarkan pencacah (2)



- Penggunaan
 - Pencacah harus bertipe ordinal, contoh: integer
 - Diketahui dengan pasti nilai awal dan nilai akhir pencacah
 - Di Python: notasi **for ... in range (...)**: dapat digunakan dalam berbagai variasi, silakan dipelajari sendiri



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



Catatan

- 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 Skema Pengulangan (1)

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

traversal

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

repeat n
times



Latihan Skema Pengulangan (2)

- 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.





Latihan Skema Pengulangan (3)

- 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



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



Studi Kasus

- Studi Kasus 1
 - Buatlah algoritma yang membaca sebuah bilangan bulat N , menuliskan: $1, 2, 3, \dots, N$ dan menjumlahkan $1 + 2 + 3 + \dots + N$ serta menuliskan hasil penjumlahan.
- Studi Kasus 2
 - Buatlah algoritma yang membaca bilangan sampai dibaca 9999, menghitung penjumlahannya dan menampilkannya ke layar
- Studi Kasus 3
 - Buatlah algoritma yang membaca N buah bilangan, menghitung penjumlahannya dan menampilkannya ke layar. Asumsi $N > 0$.

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
```

Contoh Pemrosesan Sekuensial Dengan MARK (1)

Studi Kasus 1



Program SUMNBil1

{ 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 \leftarrow 0 { Inisialisasi }
i \leftarrow 1 { First Element }
while (i \leq N) do { EOP: i > N }
 output(i) { Proses current element }
 Sum \leftarrow Sum + i { Proses current element }
 i \leftarrow i + 1 { Next Elmt }
{ i > N, i = N + 1, Sum = 1 + 2 + 3 + ... + N }
output(Sum) { Terminasi }

Contoh Pemrosesan Sekuensial Dengan MARK (2)

Studi Kasus 2



Program SumBill

{ Input beberapa integer s.d. dimasukkan 9999, menghitung penjumlahan semua bilangan, dan menampilkannya ke layar.

Model dengan mark; tanpa penanganan kasus kosong }

KAMUS

a : integer { read integer dari pengguna }
Sum : integer { jumlah }
i : integer { banyaknya bilangan yg. diinput user }

ALGORITMA

Sum ← 0	{ Inisialisasi }
<u>input</u> (a)	{ First Element }
<u>while</u> (a ≠ 9999) <u>do</u>	{ EOP: a = 9999 }
Sum ← Sum + a	{ Proses current element }
<u>input</u> (a)	{ Next Elmt }
{ a = 999 }	
<u>output</u> (Sum)	{ 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
```

Contoh Pemrosesan Sekuensial Dengan MARK (3)

Studi Kasus 1



Program SUMNBil2

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

Model dengan mark, dengan penanganan kasus kosong }

KAMUS

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

ALGORITMA

input(N) { Inisialisasi }
i \leftarrow 1 { First Element }
if (i > N) then { EOP : i > N }
 output("Data kosong")
else { i \leq N }
 Sum \leftarrow 0 { Inisialisasi }
 repeat
 output(i) { Proses current element }
 Sum \leftarrow Sum + i { Proses current element }
 i \leftarrow i + 1 { Next Elmt }
 until (i > N) { EOP : i > N }
 output(Sum) { Terminasi }

Contoh Pemrosesan Sekuensial Dengan MARK (4)

Studi Kasus 2



Program SUMBil2

{ Input beberapa integer s.d. dimasukkan 9999, menghitung penjumlahan semua bilangan, dan menampilkan ke layar.

Model dengan mark, dengan penanganan kasus kosong secara khusus }

KAMUS

a : integer { read integer dari pengguna }
Sum : integer { jumlah }
i : integer { banyaknya bilangan yg. diinput user }

ALGORITMA

<u>input</u> (a)	{ First Element }
<u>if</u> (a = 9999) <u>then</u>	{ EOP: a = 9999 }
<u>output</u> ("Data kosong")	{ Proses Kasus Kosong }
<u>else</u> { a ≠ 9999 }	
Sum ← 0	{ Inisialisasi }
<u>repeat</u>	
Sum ← Sum + a	{ Proses current element }
<u>input</u> (a)	{ Next Elmt }
<u>until</u> (a = 9999)	{ EOP: a = 9999 }
<u>output</u> (Sum)	{ 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

Contoh Pemrosesan Sekuensial Tanpa MARK (1)

Studi Kasus 1



Program SUMNBil3

{ Menjumlahkan $1+2+3+\dots+N$ dengan N dibaca. **Asumsi $N > 0$.**

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 }
{ i = N, Sum = 1 + 2 + 3 + ... + N }
output(Sum) { Terminasi }

Contoh Pemrosesan Sekuensial Tanpa MARK (1)

Studi Kasus 1



Program SUMNBil3

{ Menjumlahkan $1+2+3+\dots+N$ dengan N dibaca. **Asumsi $N > 0$.**

Model tanpa mark }

KAMUS

i : integer { bilangan yang akan dijumlahkan }
 N : integer > 0 { banyaknya bilangan yang akan 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 }
{ $i = N, Sum = 1 + 2 + 3 + \dots + N$ }
output(Sum) { Terminasi }

N harus > 0 ; Jika tidak, maka jika $N \leq 0$, dimungkinkan terjadi "kebocoran"

Contoh Pemrosesan Sekuensial Tanpa MARK (2)

Studi Kasus 3



Program SUMBi13

{ Input bilangan N, lalu dibaca N buah integer, menjumlahkan semua integer, dan menampilkan hasilnya ke layar. Asumsi $N > 0$. **Model tanpa mark dengan iterate stop** }

KAMUS

N : integer { banyaknya bilangan, $N > 0$ }
a : integer { input integer dari pengguna }
Sum : integer { jumlah }
i : integer { counter }

ALGORITMA

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

Contoh Pemrosesan Sekuensial Tanpa MARK (3)

Studi Kasus 3



Program SUMBil3

{ Program yg meminta input bilangan N, lalu meminta input integer sebanyak N, menjumlahkannya dan menampilkan hasilnya ke layar. Asumsi $N > 0$. **Skema tanpa mark dengan loop traversal** }

Kamus

N : integer { banyaknya bilangan, $N > 0$ }
a : integer { input integer dari pengguna }
Sum : integer { jumlah }
i : integer { banyaknya bilangan yg dibaca user }

Algoritma

<u>input</u> (N)	{ Inisialisasi }	Versi traversal, tanpa mark
Sum \leftarrow 0	{ Inisialisasi }	
<u>i traversal</u> [1..N]	{ First Elmt, Next Elmt, EOP }	
<u>input</u> (a)	{ Proses current element }	
Sum \leftarrow Sum + a	{ Proses current element }	
<u>output</u> (Sum)	{ Terminasi }	

Skema Pemrosesan Sekuensial Tanpa MARK (3)



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
```

Contoh Pemrosesan Sekuensial Tanpa MARK (4)



Program Bilangan

```
{ Menerima masukan sejumlah bilangan; menuliskan apakah  
negatif, positif, atau nol untuk tiap bilangan; dan  
menghitung jumlah semua bilangan.  
Pembacaan dihentikan jika bilangan = 0. }
```

{ Skema tanpa mark }

KAMUS

```
bil : integer          { input integer dari pengguna }  
Sum : integer          { jumlah }
```

ALGORITMA

```
Sum  $\leftarrow$  0                { Inisialisasi }  
repeat  
    input(bil)                { Next Elmt (tmsk. First Elmt) }  
    depend on (bil)           { Proses current element }  
        bil > 0 : output("Positif")  
        bil < 0 : output("Negatif")  
        bil = 0 : output("Nol")  
    Sum  $\leftarrow$  Sum + bil      { Proses current element }  
until (bil = 0)              { EOP: bil = 0 }  
output(Sum)                  { Terminasi }
```

Contoh Pemrosesan Sekuensial Tanpa MARK (4)



Program Bilangan

```
{ Menerima masukan sejumlah bilangan; menuliskan apakah  
negatif, positif, atau nol untuk tiap bilangan; dan  
menghitung jumlah semua bilangan.  
Pembacaan dihentikan jika bilangan = 0. }
```

{ Skema tanpa mark }

KAMUS

```
bil : integer          { input integer dari pengguna }  
Sum : integer          { jumlah }
```

ALGORITMA

```
Sum ← 0                      { Inisialisasi }  
repeat  
    input(bil)                { Next Elmt (tmsk  
    depend on (bil)           { Proses current  
        bil > 0 : output("Positif")  
        bil < 0 : output("Negatif")  
        bil = 0 : output("Nol")  
    Sum ← Sum + bil           { Proses current element }  
until (bil = 0)              { EOP: bil = 0 }  
output(Sum)                  { Terminasi }
```

bil = 0 tetap diproses;
namun juga menjadi
EOP

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 apa pun dapat dipakai, tapi ingat *nature* dari masing-masing skema pengulangan



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>