PENGULANGAN

Bagian 1 : Notasi

Tim Pengajar KU1071

Sem. 1 2009-2010

Tujuan

- Mahasiswa memahami jenis-jenis pengulangan dan penggunaannya serta memahami elemen-elemen dalam pengulangan.
- Mahasiswa dapat menggunakan notasi pengulangan yang sesuai dengan benar
- Mahasiswa memahami skema pengulangan dan penggunaannya.
- Mahasiswa dapat memanfaatkan jenis-jenis pengulangan dengan tepat dalam menyelesaikan persoalan sederhana yang diberikan.

Pengulangan: Latar Belakang

- **Melakukan** suatu **instruksi**, bahkan **aksi**, secara **berulang-ulang**
  - Komputer: memiliki performansi yang sama
  - Manusia: punya kecenderungan untuk melakukan kesalahan (karena letih atau bosan)

Pengulangan

- Elemen:
  - **Kondisi pengulangan berhenti**: ekspresi lojik
  - **Badan pengulangan**: aksi yang diulang
- Notasi pengulangan:
  1. Berdasarkan jumlah pengulangan
  2. Berdasarkan kondisi berhenti
  3. Berdasarkan kondisi pengulangan
  4. Berdasarkan dua aksi
  5. Berdasarkan pencacah

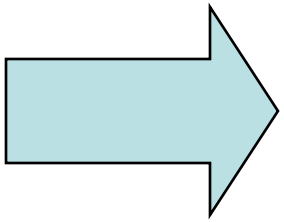
1. Berdasarkan jumlah pengulangan

```
repeat n times
    Aksi
```

- Aksi akan diulang sebanyak n kali, dan bukan urusan pemrogram untuk mengelola pengulangan tersebut
- Dengan hanya menyebutkan pengulangan tersebut, pengulangan pasti akan berhenti suatu saat

Contoh 1

```
Kupas1Kentang
Kupas1Kentang
Kupas1Kentang
Kupas1Kentang
```



```
Repeat 4 times
   Kupas1Kentang
```

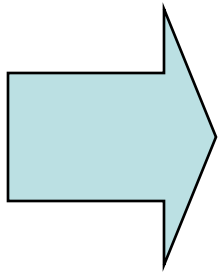
2. Pengulangan Berdasarkan Kondisi Berhenti

repeat
aksi
until *kondisi-berhenti*

- *Aksi* akan dihentikan jika *kondisi-berhenti* dipenuhi (berharga true), akan diulang jika *kondisi-berhenti* belum tercapai
- Badan pengulangan pada notasi ini (*Aksi*) minimal akan dilakukan satu kali karena pada waktu eksekusi pengulangan yang pertama tidak dilakukan test terhadap *kondisi-berhenti*
- Test terhadap kondisi berhenti dilakukan setelah *Aksi* dilaksanakan
- Pengulangan berpotensi mengalami “kebocoran”, jika ada kemungkinan bahwa seharusnya *Aksi* tidak pernah boleh dilakukan untuk kasus tertentu

Contoh 2

```
{min ada 1 kentang dlm
   kantong}
Kupas1Kentang
if kantong belum kosong then
   Kupas1Kentang
if kantong belum kosong then
   Kupas1Kentang
if kantong belum kosong then
   Kupas1Kentang
    . . .
if kantong belum kosong then
   Kupas1Kentang
```



```
repeat
     Kupas1Kentang
until kantong kosong
```

3. Pengulangan Berdasarkan Kondisi Pengulangan

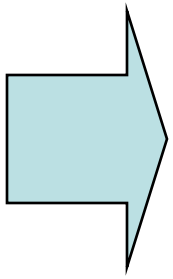
while (*kondisi-pengulangan*) **do**
aksi

{Kondisi berhenti dicapai di titik program ini}

- *Aksi* akan dilakukan selama *kondisi-pengulangan* masih dipenuhi (berharga true)
- Test terhadap *kondisi-pengulangan* dilakukan setiap kali sebelum *aksi* dilaksanakan
- Pengulangan ini berpotensi untuk menimbulkan aksi “kosong” (tidak pernah melakukan apa-apa) karena pada test yang pertama, *kondisi-pengulangan* tidak dipenuhi (berharga false)
  - Badan pengulangan (*aksi*) pada notasi ini mungkin tidak akan pernah dilakukan, karena sebelum *aksi* yang pertama dieksekusi dilakukan test terhadap *kondisi-berhenti*

Contoh 3

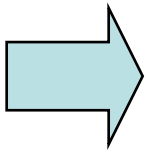
```
{kantong boleh kosong}
if kantong tidak kosong then
   Kupas1Kentang
if kantong tidak kosong then
   Kupas1Kentang
if kantong tidak kosong then
   Kupas1Kentang
    . . .
if kantong tidak kosong then
   Kupas1Kentang
```



While *kantong tidak kosong* **do**
    Kupas1Kentang
{kantong kosong}

Contoh 4

```
{min ada 1 kentang dlm
   kantong}
Kupas1Kentang
if kantong tidak kosong then
   Kupas1Kentang
if kantong tidak kosong then
   Kupas1Kentang
if kantong tidak kosong then
   Kupas1Kentang
    . . .
if kantong tidak kosong then
   Kupas1Kentang
```



```
Kupas1Kentang
While kantong tdk kosong do
    Kupas1Kentang
{kantong kosong}
```

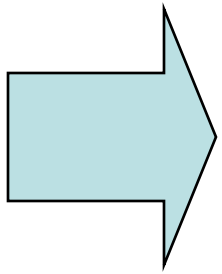
4. Berdasarkan dua aksi

```
Iterate
       Aksi-1
Stop (kondisi berhenti
       Aksi-2
{kondisi berhenti dicapai di titik program ini}
```

- Pengulangan ini seolah-olah adalah “gabungan” antara bentuk pengulangan kedua dan ketiga
- Mekanisme yang dilakukan oleh pengulangan ini adalah dengan melakukan secara otomatis Aksi-1 pada eksekusi yang pertama kemudian dilakukan test terhadap kondisi berhenti
- Tergantung kepada kondisi berhenti yang ditest:
  - Aksi-2 akan diaktifkan dan kemudian aksi-1 yang berikutnya diulang, atau
  - Pengulangan dihentikan karena efek neto dari aksi-1 menghasilkan kondisi berhenti
- Pengulangan ini berguna untuk kasus-kasus di mana Aksi-2 merupakan hal yang harus dilakukan tergantung dari hasil *Aksi-1*.

Contoh 5

```
{min ada 1 kentang yg
   dipegang}
Kupas1Kentang
if kantong tidak kosong then
   Ambil1Kentang
   Kupas1Kentang
if kantong tidak kosong then
   Ambil1Kentang
   Kupas1Kentang
    . . .
if kantong tidak kosong then
   Ambil1Kentang
   Kupas1Kentang
```



```
iterate
     Kupas1Kentang
Stop kantong kosong
     Ambil1Kentang
```

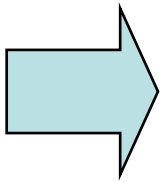
5. Pengulangan Berdasarkan Pencacah

nama-pencacah **traversal** [*range-harga*]
  aksi

- *nama-pencacah* harus suatu type yang terdefinisi suksesor dan predesesornya, setelah pelaksanaan pengulangan selesai, harga yang tersimpan pada nama-pencacah tidak terdefinisi: jika hendak dipakai, harus didefinisikan kembali
- Aksi akan dilakukan dengan memperhitungkan harga-harga dari nama-pencacah yang di-”jelajahi”
- Dengan memakai pengulangan ini, pemrogram tidak perlu melakukan operasi terhadap suksesor/predesesor karena setiap kali selesai melakukan *Aksi*, otomatis mesin akan melakukan operasi untuk mendapatkan suksesor dari harga yang berlaku saat itu untuk nama
- *range-harga* bisa dari kecil ke besar atau sebaliknya

Contoh 6

```
KupasKentang1
KupasKentang2
KupasKentang3
KupasKentang4
```



```
i traversal [1..4]
   KupasKentang(i)
```

Berapa Hello di layar?

```
Stop ← not(TRUE)
Repeat
   output('Hello')
Until stop

Repeat
   output('Hello')
Until TRUE

Repeat
   output('Hello')
Until FALSE
```

Berapa Hello di layar? (2)

```
i←1
While (i<5) do
    output('Hello')

While FALSE do
    output('Hello')

While TRUE do
    output('Hello')
```

Contoh Penggunaan

- Deskripsi Persoalan:
  - Tuliskanlah sebuah program yang membaca sebuah nilai N (integer positif lebih besar nol), dan menuliskan output nilai 1,2,3,4, … s.d. N berderet ke bawah sbb

    ```
    1
    2
    3
    …
    N
    ```
  - Contoh:
    - Jika N = 3, maka outputnya adalah

      ```
      1
      2
      3
      ```
    - Jika N = 1, maka outputnya adalah

      ```
      1
      ```

Program TULISBIL2

Program TULISBIL2
{dibaca N ≥ 0, menuliskan 1,2,3, … N berderet ke bawah, dengan bentuk repeat … until …}

Kamus
  i,N : integer {bilangan yang akan ditulis}

Algoritma:

```
input (N)
i ← 1
repeat
  output (i)
  i ← i + 1
until (i>N)
```

Catatan:

1. Dengan bentuk ini, **harus dijamin** bahwa nilai N yang dibaca sesuai dengan spesifikasi, yaitu **N ≥ 0**. Jika nilai yang diberikan tidak sesuai dengan spesifikasi maka akan tertulis angka 1. Untuk mengatasi hal ini dapat dilakukan misalnya bahwa nilai N yang dibaca “diulang” sehingga memenuhi persyaratan N ≥ 0.
2. Nama yang didefinisikan sebagai i akan bernilai 1 .. N+1.
3. Anda dapat mendefinisikan i sebagai bilangan yang **sudah** ditulis dan memulai i dengan 0. Dalam hal ini nilai i adalah 0 .. N.

Program TULISBIL3

Program TULISBIL3
{dibaca N ≥ 0, menuliskan 1,2,3, … N berderet ke bawah, dengan bentuk while .. do … }

Kamus
i,N : integer {bilangan yang akan ditulis}

Algoritma :

```
input (N)
i ← 1
while i ≤ N do
   output (i)
   i ← i + 1
{i > N}
```

Catatan:

1. Dengan bentuk ini, nilai 1 belum tentu ditulis. Jika ternyata pengguna memberikan nilai N yang negatif, program tidak menuliskan apa-apa karena kondisi yang diperiksa pertama kali (i ≤ N) menghasilkan false.
2. Nama yang didefinisikan sebagai i akan bernilai 1 .. N+1.
3. Anda dapat mendefinisikan i sebagai bilangan yang **sudah** ditulis dan memulai i dengan 0. Dalam hal ini nilai i adalah 0 .. N.

Program TULISBIL5

Program TULISBIL5
{dibaca N ≥ 0, menuliskan 1,2,3, … N berderet ke bawah, dengan bentuk iterate

Kamus
  i,N : integer {bilangan yang akan ditulis}

Algoritma:
```
input (N)
 i traversal [1..N]
   output (i)
```

Penjelasan:

1. Dengan bentuk ini ekivalen dengan bentuk iterate, untuk i [1..N], namun pertambahan nilai i ditangani secara implisit oleh pemroses bahasa
2. Jika N yang diberikan oleh pengguna bernilai negatif, maka tidak akan terjadi aksi penulisan karena nilai i di luar domain [1..N]

Translasi ke Pascal (1)

Algoritma	Pascal

while <kondisi-ULANG> do
  Aksi
	while (kondisi-ULANG) **do**
	begin
	  Aksi
	end;

repeat
  Aksi
until <kondisi-STOP>
	repeat
	  Aksi
	until (kondisi-STOP);

Translasi ke Pascal (2)

Algoritma	Pascal

iterate
  Aksi-A
stop <kondisi-STOP>
  Aksi-B
	(* deklarasi stop : boolean *)
	stop := false;
	repeat
	  Aksi-A;
	  if (kondisi-STOP) **then**
	  stop := true
	  else
	  Aksi-B;
	until (stop);

Translasi ke Pascal (3)

Algoritma	Pascal

i traversal [Awal..Akhir]
Aksi
	/* Jika Awal <= Akhir */
	for (i := Awal **to** Akhir)
	do
	begin
	Aksi
	end;

	/* Jika Awal >= Akhir */
	for (i := Awal **downto** Akhir) **do**
	begin
	Aksi
	end;