BAHASA PEMROGRAMAN *BASIC*

OBJEKTIF:

1. Mahasiswa dapat memahami kondisi perulangan `for-next`.
2. Mahasiswa dapat memahami kondisi perulangan `for-next (nested loop)`.
3. Mahasiswa dapat memahami kondisi perulangan `do-while-loop`.
4. Mahasiswa dapat memahami kondisi perulangan `do-until-loop`.
5. Mahasiswa mampu mengenali perbedaan penggunaan kondisi perulangan `for-next`, `for-next` (nested loop), `do-while-loop`, dan `do-until-loop`.

3.1 Struktur Kendali Perulangan

Seperti yang sudah dibahas pada bab sebelumnya, struktur kendali memegang peranan yang sangat penting dalam sebuah program karena dapat mengatur bagaimana aliran program berdasarkan kondisi yang terpenuhi. Selain struktur kendali kondisi, pada bahasa pemrograman *basic* juga menyediakan sebuah struktur kendali perulangan. Perulangan adalah proses mengulang suatu pernyataan sampai kondisi tertentu terpenuhi dan proses perulangan dikendalikan oleh pemrogram. Dengan menggunakan proses perulangan, penulisan kode program dapat menjadi lebih sederhana dan efisien.

Basic memiliki tiga jenis perulangan diantaranya perulangan `for-next`, perulangan `do-while-loop` dan perulangan `do-until-loop`.

3.1.1 Perulangan `for - next`

Perulangan `for-next` merupakan struktur yang mengulang suatu blok pernyataan dengan jumlah perulangannya ditentukan berdasarkan perhitungannya (sistem *counter*).

Bentuk umum perulangan `for-next`:

```
FOR iterator = startvalue TO endvalue [STEP stepvalue]
[ statement block ]
NEXT [ iterator ]
```

Penjelasannya:

- *iterator* disebut juga sebagai variabel pencacah yang hanya menyimpan sebuah nilai numerik.
- *startvalue* (nilai awal) dan *endvalue* (nilai akhir) menjadi kondisi pada perulangan.
- *stepvalue* merupakan penambahan nilai variabel pencacah pada setiap perulangan. Apabila *stepvalue* tidak di tulis maka secara default dianggap bernilai 1.
- *statement block* merupakan blok pernyataan yang di eksekusi program.

Contoh program menggunakan perulangan `for-next`:

```
DIM y as integer
FOR y = 1 TO 5
PRINT y
NEXT y
END
```

Output program:

```
1
2
3
4
5
```

Pada contoh program diatas, dideklarasikan variabel `y` bertipe data integer. Kemudian dilakukan perulangan terhadap variabel `y` dengan nilai awal perulangan sama dengan satu dan nilai akhir perulangan sama dengan lima. Jadi perulangan akan dilakukan sebanyak lima kali sesuai kondisi yang ditentukan. Dimana setiap perulangan akan dilakukan pencetakan nilai pada variabel `y` ke *compiler* dan perulangan akan berhenti ketika nilai pada variabel `y` menyimpan nilai akhir perulangan yaitu sama dengan lima.

3.1.2 Perulangan `for-next` (Nested Loop)

Perulangan `for-next` (nested loop) dilakukan jika terdapat dua buah perulangan. Jadi, nested loop (perulangan bersarang) memuat sebuah perulangan yang berada di dalam perulangan lainnya.

Bentuk umum perulangan `for-next` (nested loop):

```
FOR iterator1 = startvalue TO endvalue [ STEP stepvalue ]
[ statement block ]
   FOR iterator2 = startvalue TO endvalue [ STEP stepvalue ]
   [ statement block ]
   NEXT [ iterator2 ]
   [ statement block ]
NEXT [ iterator1 ]
```

Perulangan `for-next` (nested loop) memiliki perulangan bagian luar dan perulangan bagian dalam. Dimana ketika program dijalankan, perulangan bagian dalam yang akan dieksekusi terlebih dahulu.

Contoh program menggunakan perulangan `for-next` (nested loop):

```
DIM as integer y, x
FOR y = 1 TO 5
    FOR x = 1 TO 5
       PRINT y;
    NEXT x
    PRINT
NEXT y
END
```

Output program:

```
11111
22222
33333
44444
55555
```

Pada contoh program diatas di deklarasikan variabel `y` dan `x` dengan masing-masing bertipe data integer. Kemudian dilakukan perulangan terhadap variabel `y` (perulangan bagian luar) dengan nilai awal perulangan sama dengan satu sampai dengan nilai akhir perulangan sama dengan lima. Dimana pada setiap perulangan terhadap variabel `y` dilakukan, maka perulangan terhadap variabel `x` (perulangan bagian dalam) akan di eksekusi dengan nilai awal kondisi perulangan sama dengan satu sampai dengan nilai akhir perulangan sama dengan lima.

Dan perulangan ini akan terus berlanjut sampai dengan perulangan terhadap variabel `y` telah mencapai nilai akhir kondisi dan pada perulangan bagian dalam telah di eksekusi sebanyak 5 kali.

3.1.3 Perulangan `do-while-loop`

Perulangan `do-while-loop` merupakan perulangan yang akan berhenti ketika kondisi bernilai `False`, dan akan terus melakukan perulangan selama kondisi bernilai `True`.

Bentuk umum perulangan *do-while-loop* 1:

```
DO WHILE [ condition ]
[ statement block ]
LOOP
```

Bentuk umum perulangan *do-while-loop* 2:

```
DO
[ statement block ]
LOOP WHILE [ condition ]
```

Dari kedua bentuk perulangan diatas, terdapat perbedaan letak kondisi perulangannya. Pada bentuk pertama kondisi berada di awal perulangan, sedangkan pada bentuk kedua kondisi berada di akhir perulangan. Tidak ada perbedaan output ketika kondisi yang diberikan sama pada kedua bentuk perulangan ini, hanya saja untuk bentuk perulangan kedua akan dilakukan minimal satu kali karena pengecekan kondisi terjadi di akhir perulangan.

Contoh program menggunakan perulangan `do-while-loop` :

```
DIM hitung as integer
hitung = 1
DO WHILE hitung < 11
PRINT hitung
hitung = hitung + 1
LOOP
END
```

Output program:

```
1
2
3
4
5
6
7
8
9
10
```

Pada contoh program di atas, di deklarasikan variabel `hitung` bertipe data integer dengan nilai awal sama dengan satu. Kemudian masuk ke perulangan `do-while-loop`, selama nilai variabel `hitung` lebih kecil dari 11 maka akan dilakukan pencetakkan nilai pada variabel `hitung` ke *compiler*. Lalu diberikan penambahan nilai pada variabel `hitung` pada setiap perulangannya dan perulangan akan berhenti ketika nilai variabel hitung tidak lebih kecil dari sebelas atau kondisi bernilai `false`.

3.1.4 Perulangan `do-until-loop`

Perulangan `do-until-loop` merupakan perulangan yang akan berhenti ketika kondisi bernilai True. Jadi perulangan ini merupakan kebalikan dari perulangan `do-while-loop` dimana pada perulangan `do-while-loop` kondisi akan terus dilakukan selama kondisi bernilai `True` sedangkan pada perulangan ini kondisi akan terus dilakukan selama kondisi bernilai `False`.

Bentuk umum perulangan `do-until-loop` 1:

```
DO UNTIL [ condition ]
[ statement block ]
LOOP
```

Bentuk umum perulangan `do-until-loop` 2:

```
DO
[ statement block ]
LOOP UNTIL [ condition ]
```

Sama seperti bentuk perulangan `do-while-loop` sebelumnya, perulangan `do-until-loop` juga memiliki dua bentuk pendeklarasian. Pada bentuk pertama kondisi berada di awal perulangan dan pada bentuk kedua kondisi berada di akhir perulangan.

Contoh program menggunakan perulangan `do-until-loop`:

```
DIM hitung as integer
DO UNTIL hitung > 10
PRINT hitung
hitung = hitung + 1
LOOP
END
```

Output program:

```
0
1
2
3
4
5
6
7
8
9
10
```

Pada contoh program diatas, di deklarasikan variabel `hitung` bertipe data integer. Kemudian perulangan akan terus dilakukan sampai dengan nilai pada variabel `hitung` lebih besar atau sama dengan dari 10. Jadi selama kondisi perulangan bernilai `false`, maka dilakukan pencetakan nilai variabel `hitung` ke *compiler*, dan dilakukan penambahan nilai variabel `hitung` setiap perulangannya.

REFERENSI

[1] FreeBASIC. Diakses melalui https://www.freebasic.net/. Diakses pada 15 Oktober 2020.

[2] documentation.help. *FreeBASIC Documentation*. Diakses melalui https://documentation.help/FreeBASIC/KeyPgIfthen.html. Diakses pada 15 Oktober 2020.