3 STRUKTUR PENGENDALIAN DALAM PERANCANGAN PROGRAM

Kita memerlukan beberapa Struktur dasar untuk mengorganisir arus pengendalian Program ataupun algoritma. Tiga di antaranya, yakni struktur **BEGIN-END**, struktur keputusan (misalnya **IF-THEN-ELSE**), dan gelung atau loop (misalnya **WHILE-DO**) merupakan pembentuk dasar dari organisasi Program yang diperlukan untuk mendukung proses Pemrograman yang sistematis. Teknik ini biasa disebut sebagai **Structured Programming**, atau **Pemrograman Terstruktur**. Teknik ini didukung pula oleh beberapa struktur pengendalian lanjut, seperti gelung **REPEAT-UNTIL**, gelung berindeks, struktur aneka pilihan (cabang ganda), yang dapat dipergunakan untuk menyederhanakan proses tersebut. Bahasa Pascal kita, termasuk salah satu bahasa yang sesuai bagi Teknik Pemrograman Terstruktur tersebut. Pascal memiliki semua fasilitas Statemen yang disebutkan di atas. Sebuah gelung dapat dibiarkan prematur dengan melaksanakan suatu Statemen EXIT. Statemen tersebut tidak langsung terdapat di dalam setiap bahasa Pemrograman serba guna, tetapi Statemen seperti itu dapat disusun dengan memakai Statemen suatu bahasa tertentu. Seorang Pemrogram akan merasakan, bahwa bila Ia berpikir mengikuti struktur tersebut, akan dihasilkan Program yang terorganisir secara baik, mudah dibaca, dan mudah dimodifikasi.

Dalam situasi Pemrograman tertentu, memang kita masih dapat memakai sistem pengendalian alih jalur tak bersyarat, yakni Statemen GO TO. Dua alat penting yang diperlukan untuk membuat suatu Program sudah dibahas, yakni Diagram Alur dan Kode-Semu atau Pseudocode, telah kita bicarakan pada Bab terdahulu. Kode-Semu sering dipakai untuk menggantikan Diagram Alur. Umumnya digunakan untuk menghasilkan algoritma yang siap diisikan ke dalam kode di dalam proses perancangan Program dari atas ke bawah, atau **top-down**. Hal ini disebut penyempurnaan secara bertahap. Selain itu masih terdapat pula alat yang disebut **Tabel Keputusan**. Tabel Keputusan dipakai untuk membuat pertimbangan yang sistematis terhadap seluruh kegiatan yang harus dilaksanaan oleh Program, jika sewaktu-waktu timbul Kondisi tertentu.

3.1 KONDISI

Pelaksanaan dari dua bentuk Statemen dasar Pemrograman, yakni bentuk decision atau keputusan, dan bentuk gelung, akan membuat pelaksanaan jalur Program ditentukan oleh eksistensi *Kondisi* tertentu. Untuk memperinci Kondisi tertentu tersebut, kita dapat memakai ekspresi Logika. Di dalam kasus yang sederhana, akan terdapat Relasi di antara ekspresi Logika tersebut.

1. APAKAH SEBENARNYA SUATU KONDISI ITU ?

Suatu **Kondisi** adalah *penetapan dari suatu nilai Variabel* atau *penetapan dari ketergantungan antara nilai dari dua atau lebih Variabel.* Penetapan tersebut dapat bernilai True atau False. Nilai dari suatu Kondisi dapat diuji, dan hasilnya adalah Kondisi True atau Kondisi False. Oleh karena itu, kedua hal tersebut dinyatakan dengan mempergunakan ekspresi Logika.

CONTOH 3-1

Kita dapat menetapkan suatu Kondisi, dengan melakukan percobaan, berupa pemberian nilai kepada Variabel, misalnya :

a. Variabel integer I = 0
   Di sini jika kita berikan nilai 3 kepada I, maka Kondisi adalah False. Sedangkan jika kita berikan nilai 0 kepada I, maka Kondisi adalah True.
b. Variabel integer I lebih kecil dari Variabel Integer K yang digandakan dengan 6
   Di sini jika diberikan nilai 4 kepada I, serta nilai 2 kepada K, maka Kondisi adalah True.
c. Variabel Real X tidak lebih kecil dari Variabel Real Y,dan Variabel Logika L bernilai True.
   Andaikan kita berikan nilai 4.34 kepada X, dan nilai 3.94 kepada Y, serta L adalah True, maka Kondisi adalah True.

Kondisi tersebut di atas, dipakai untuk mencari alternatif bagi pelaksanaan jalur tugas di dalam Program, atau untuk pelaksanaan ulang sekelompok Statemen di dalam gelung.

Suatu Kondisi sederhana merupakan suatu Relasi, contohnya adalah Kondisi pada contoh 3.1 (a), dan (b). Sedangkan yang disebut Kondisi Majemuk adalah Kondisi yang merupakan suatu ekspresi Logika, terdiri dari beberapa operand berupa Kondisi sederhana, dan beberapa Operator berupa Operator Logika seperti AND, OR, ataupun NOT.

Berhubung suatu Kondisi adalah dinyatakan sebagai ekspresi Logika, maka nilainya dapat disimpan di dalam Variabel Logika sebagai nilai True atau False.

2. OPERATOR RELASIONAL

Suatu **Kondisi sederhana** adalah Relâsi, yang merupakan suatu ekspresi Logika, yang terdiri dari dua ekspresi Aritmetika. Kedua ekspresi Aritmetika tersebut dihubungkan oleh Operator Relasional, yakni salah satu dari tanda

<	lebih kecil
<=	lebih kecil atau sama dengan
=	sama dengan
<>	tidak sama dengan
>	lebih besar
>=	lebih besar atau sama dengan

CONTOH 3-2

Kondisi yang dikemukakan di dalam contoh 3-1 dapat ditulis sebagai suatu Relasi berikut ini:

a. I = 0
b. I < 6 * K
c. Kondisi tersebut bukan merupakan Kondisi sederhana, jadi tak dapat kita sajikan sebagai Relasi biasa. Ia adalah suatu Kondisi Majemuk,

Relasi tersebut menghasilkan nilai True, jika kondisinya memang demikian (Kondisi benar). Jika tidak, nilainya adalah False. Perhatikan bahwa nilai tersebut tidak diperintahkan untuk memasuki salah satu alokasi. Ia hanya diuji saja.

Untuk menguji kebenaran dari suatu Relasi, ekspresi pada ruas kiri maupun ruas kanan kita evaluasi nilainya. Kemudian dengan menjawab pertanyaan : "apakah Operator Relasional memberi jawaban yang betul ?" Jika ya, maka tes tersebut menghasilkan nilai True, jika tidak maka False.

CONTOH 3-3

Jika pada suatu waktu, Relasi

$$A * B + C >= D - E$$

dievaluasi, dan Variabel diasumsikan berjenis integer, dengan nilai Variabel adalah sebagai berikut :

$$A=1 \quad B=2 \quad C=3 \quad D=10 \quad E=5$$

kemudian jika Relasi tersebut diuji, akan didapat

$$A * B + C = 5 ; D - E = 5;$$

maka Relasi atau Kondisi sederhana tersebut menghasilkan nilai True.

3. KONDISI MAJEMUK

Suatu Kondisi Majemuk dinyatakan oleh suatu *ekspresi Logika*. Suatu Kondisi Majemuk mengandung:

(a) Operator Aritmetika :

** (Operator pangkat, kadang-kadang diberi tanda ^)
* /
+ -
tanda tersebut hanya dapat digunakan untuk Variabel dan Konstanta numerik.

(b) Operator Relasional :

<, <=, =, =, >, >=

tanda tersebut hanya dapat digunakan untuk data numerik.

(c) Operator Logika (Boolean):

NOT (bukan)
AND (dan)
OR (atau)

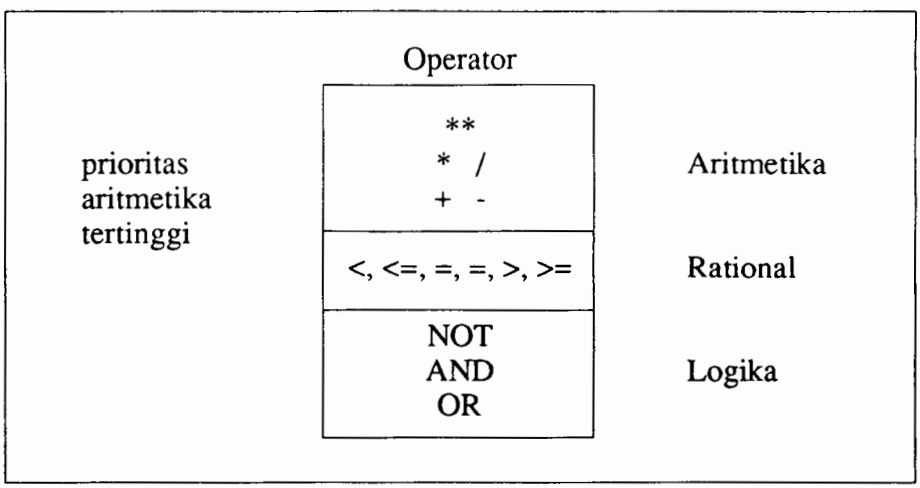
Hasil dari penggunaan Operator Boolean, diperlihatkan oleh nilai kebenaran di dalam Tabel 3-1. Kalkulus dari Variabel Logika disebut **aljabar Boole.**

Tabel 3-1. Tabel nilai bagi Operator Logika
P dan Q merupakan Variabel Logika.

NILAI VARIABEL		HASIL OPERASI		
P	Q	NOT P	P AND Q	P OR Q
False	False	True	False	False
False	True	True	False	True
True	False	False	False	True
True	True	False	True	True

Operator Logika hanya dapat beroperasi pada Variabel Logika atau Konstanta Logika (True dan False).

Prioritas operasional yang berlaku di dalam ekspresi Logika digabungkan di dalam gambar 3-1.

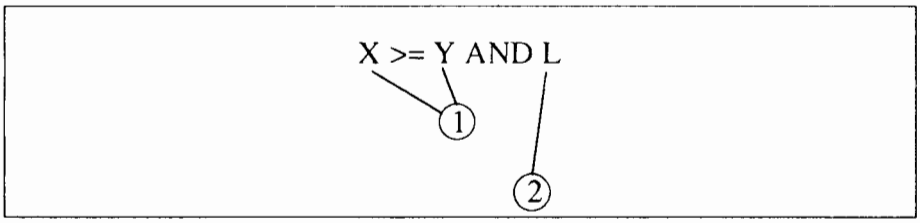


GAMBAR 3-1 Prioritas Operator di dalam ekspresi Logika.

Prioritas tersebut dapat diubah dengan memberi tanda kurung. Sistem pelaksanaan dilaksanakan dari kiri ke kanan.

CONTOH 3-4.

Untuk mempermudah perumusan Statemen di dalam contoh 3-1 (c), kita dapat memakai Kondisi Majemuk sebagai berikut :



GAMBAR 3-2

perhatikan bagaimana bagian kedua dari rumus tersebut dilaksanakan.

CONTOH 3-5

Kondisi di bawah ini :

JUMLAH lebih besar dari MODAL tetapi tidak lebih besar dari selisih antara PENJUALAN dan MODAL

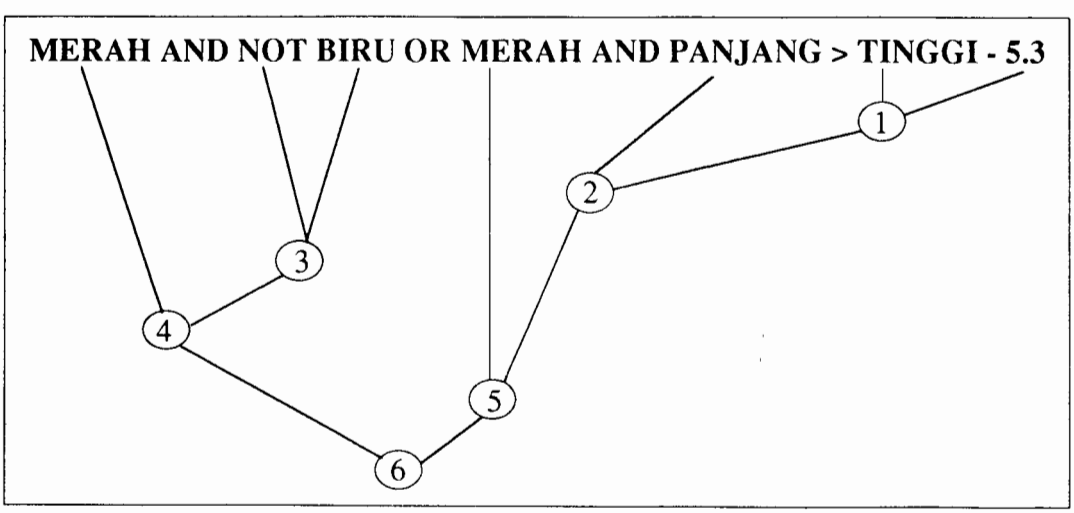
(Di sini ketiga besaran tersebut merupakan Variabel Real)

dapat kita rumuskan sebagai :

JUMLAH > MODAL AND JUMLAH <= PENJUALAN - MODAL

CONTOH 3-6

Gambar 3-3 memberikan gambaran tentang cara evaluasi suatu Kondisi Majemuk. Di sini MERAH dan BIRU merupakan Variabel Logika dan PANJANG serta DALAM merupakan Variabel Real.



GAMBAR 3.3

Catatan :

1. Karena suatu Relasi mengevaluasi nilai Logika, maka dilakukan penggabungan dengan Konstanta Logika lainnya melalui Operator Logika (seperti dalam langkah ke-5).

2. Batas antara **AND** dan **NOT** (serta **OR** dan **NOT**) tidak melanggar sintaks dari ekspresi Logika; operasi dari **AND** dan **OR** tidak dapat dibatasi secara mutlak di dalam ekspresi Logika.

3.2 KEPUTUSAN

Jika Pemrogram ingin menetapkan dua alternatif jalur pelaksanaan di dalam suatu algoritma, atau Program, maka pilihan yang diambilnya harus berdasarkan pada adanya Kondisi tertentu.

Pengambilan jalur seperti ini disebut pengambilan alih jalur bersyarat. Terdapatnya kombinasi dari beberapa Kondisi (ekspresi dari kondisi) menyebabkan terdapatnya beberapa alternatif keputusan.

Untuk dapat memilih beberapa alternatif tersebut, keputusan ditempatkan secara bersusun dan berurutan. Ada suatu alat yang disebut "**desicion table**" atau Tabel Keputusan, yang dapat membantu Pemrogram dalam penyususnan perancangan Program jika menghadapi berbagai Kondisi yang rumit, serta dalam mempertimbangkan tindakan yang harus dilakukannya.

1. KEPUTUSAN BERANGKAI IF-THEN-ELSE

Statemen keputusan yang berjalur "cabang" merupakan suatu mekanisme untuk menetapkan dua pilihan yang harus dipilih salah satu cabangnya, guna dilaksanakan sebagai tindakan dari Statemen keputusan itu. Keadaan tersebut memungkinkan kita untuk memilih :"apabila terjadi suatu Kondisi tertentu, dilaksanakan salah satu pilihan, jika tidak terjadi Kondisi seperti itu, pilihan lain yang dikerjakan."

Kondisi dinyatakan dengan ekspresi Logika. Bentuk umum dari Statemen keputusan ialah:

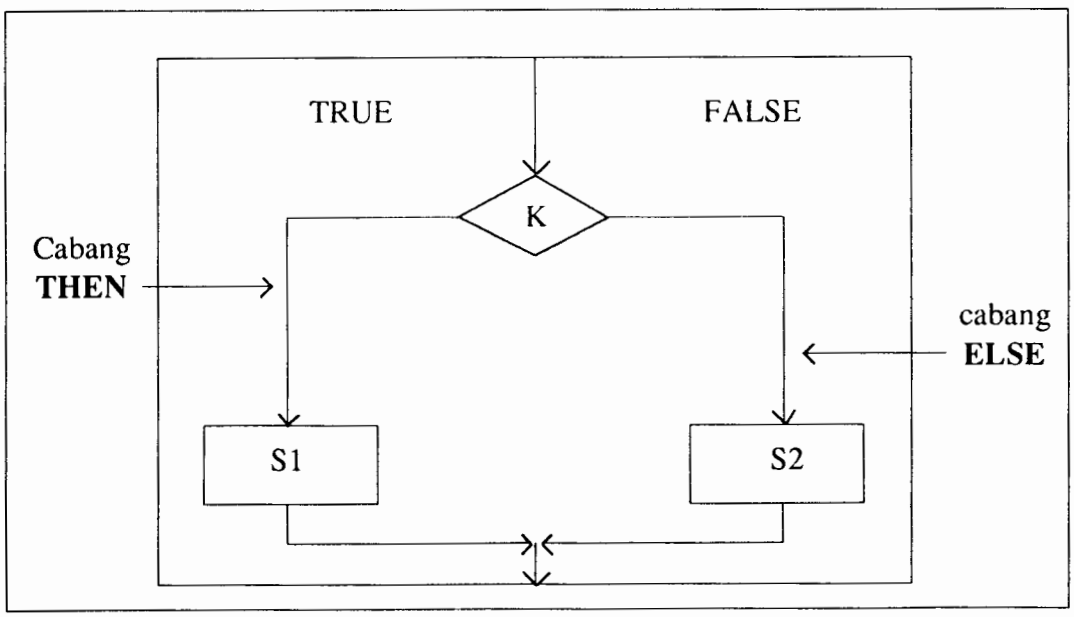
IF K **THEN** S1 **ELSE** S2

S1 dan S2 merupakan *Statemen* (atau S*ekelompok Statemen / Statemen Majemuk*) dan K merupakan suatu *Kondisi*.

Statemen keputusan dilaksanakan sebagai berikut :

a. Kondisi dicoba, dan akan menghasilkan nilai **True** atau **False**;
b. Jika nilainya adalah **True** maka S1 dilaksanakan; jika nilainya **False**, maka dilaksanakan S2 (**ELSE**)
   Selanjutnya dilaksanakan Statemen sesudah S2.

Diagram alur yang memakai bentuk Statemen IF-THEN-ELSE diperlihatkan pada gambar 3-4.



GAMBAR 3-4 Konstruksi**IF-THEN-ELSE**

CONTOH 3-7

Jawaban terhadap problema di bawah ini akan dikembangkan terus dalam buku ini.

Problema:

Pandang bahwa kita mempuntai sebanyak N buah Integer, yang akan dibaca secara berturut-turut oleh Komputer.Komputer diminta untuk menentukan berapa banyak Integer Positif serta Nol, dan berapa banyak Integer Negatif.

Untuk memproses setiap integer tertentu yang sedang dibaca, Statemen berikut ini agar dimasukan ke dalam algoritma kita.

IF integer lebih besar atau sama dengan 0 **THEN** tambahkan 1 pada kounter integer positif **ELSE** tambahkan 1 pada kounter integer negatif.

PELAKSANAAN IF-THEN-ELSE DALAM PASCAL

Semua pembicaraan kita tentang Keputusan, Kondisi, Ekspresi Logika ataupun tentang Statemen **IF-THEN-ELSE** yang lalu, adalah berlaku dengan baik dalam Pemrograman Pascal. Berikut ini beberapa contoh Program Pascal yang mengandung bentuk Keputusan.

```
PROGRAM Lulus _ Tidak;
CONST
    Batas      = 60.00;
VAR
    N1,N2      : Integer;
    Nrata      : Real;
BEGIN
    Read (N1,N2);
    Nrata      = (N1+N2)/2;
    IF (Nrata >= Batas) THEN
        WRITELN ('Lulus')
    ELSE
        WRITELN ('Tidak Lulus')
END.
```

```
PROGRAM Upah _ Mingguan;
CONST
    Batas     = 40;
    Biasa     = 3.50;
    Khusus    = 5.00;
VAR
    Jam _ Kerja :  Integer;
    Upah        :  Real;
BEGIN
    Read (Jam_Kerja);
    IF (Jam_Kerja <= Batas) THEN
            Upah   :=   Jam_ Kerja * Biasa
    ELSE
            Upah   :=   (Jam _ Kerja - Batas) * Khusus + 200.00;
    WRITELN ('Besarnya upah = ', Upah)
END .
```

```
PROGRAM Modulo;
CONST
    Batas            =      40;
    Harga_Satuan     =    1500;
    Harga_Kotakan    =   14000;
VAR
    Pembelian        :  Integer;
    Jumlah_Satuan    :  Integer;
    Jumlah_Kotakan   :  Integer;
    Harga            :  Integer;
BEGIN
    Read (Pembelian);
    IF (Pembelian < Batas) THEN
```

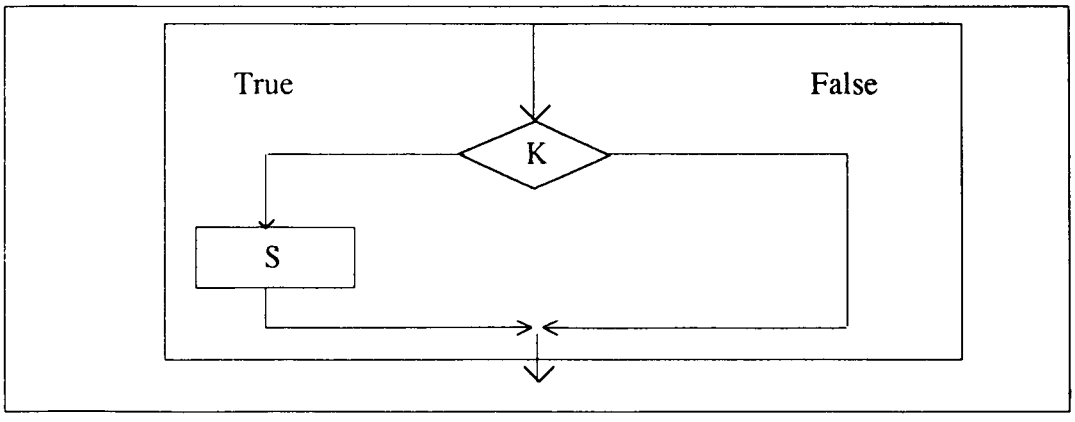
```
        BEGIN
            Jumlah _ Satuan   :=  Pembelian;
            Jumlah _ Kotakan  :=  0
        END;
    ELSE

        BEGIN
            Jumlah _ Satuan   :=  Pembelian MOD Batas;
            Jumlah _ Kotakan  :=  Pembelian DIV Batas;
            Pembayaran        :=  Jumlah_Satuan * Harga _ Satuan
                                  + Jumlah _ Kotakan * Harga _ Kotakan;
        END;
    WRITELN     ('Besarnya pembayaran = ', Pembayaran);
    WRITELN     (' untuk ',Jumlah_Satuan,' buah dan',
                 Jumlah _ Kotakan,' kotak')
END .
```

KEPUTUSAN BERANGKAI IF-THEN

Suatu bentuk lain Statemen Keputusan adalah bentuk **IF-THEN**. Statemen seperti itu memungkinkan pilihan : "jika terjadi suatu Kondisi tertentu, laksanakan Statemen S; jika tidak, laksanakan Statemen di bawah S di dalam Program."

Diagram Alur dari bentuk Statemen **IF-THEN** diperlihatkan pada gambar 3-5



Gambar 3-5 Konstruksi**IF-THEN**

Untuk tidak membingungkan kita, Statemen tersebut dapat disusun sebagai berikut :

IF K THEN
S
ELSE

di sini jalur **ELSE** adalah hampa.

Ketidak jelasan mungkin terjadi, apabila **ELSE** dihilangkan dalam rangkaian keputusan yang bersusun. Di dalam beberapa bahasa Pemrograman, cabang jalur **ELSE** dapat dihilangkan karena **ELSE** selalu berkaitan dengan **IF** tanpa perlu diikuti oleh **ELSE**-nya sendiri.
Bahasa Pascal, seperti halnya BASIC, memperbolehkan pula penggunaan Statemen **IF-THEN** tersebut. Berikut ini beberapa Program yang menggunakan Statemen tersebut.

```
PROGRAM Lulus _ Ujian;
CONST
    Batas   = 60.00;
VAR
    N1,N2   :  Integer;
    Nrata   :  Real;
BEGIN
    Read (N1,N2);
    Nrata   := (N1+N2)/2;
    IF (Nrata >= Batas) THEN
        WRITELN ('Lulus')
END
```

```
PROGRAM Upah _ Lembur;
CONST
    Batas          = 40;
    Lembur         = 5.00;
VAR
    Jam _ Kerja    : Integer;
    Uang _ Lembur  : Real;

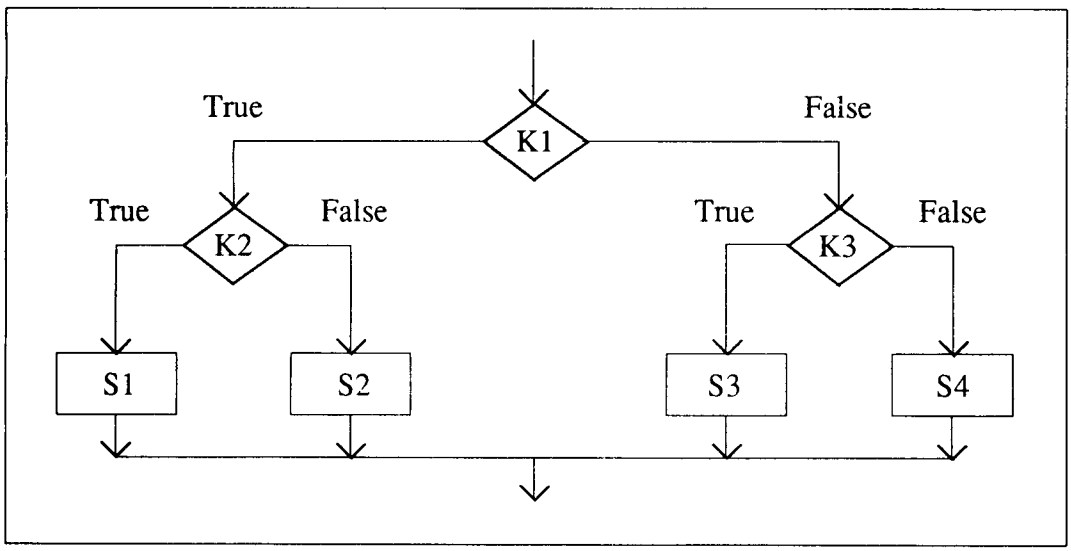
BEGIN
    Read (Jam_Kerja);
    IF (Jam_Kerja > Batas) THEN
        BEGIN
            Uang _ Lembur := (Jam _ Kerja - Batas) * Lembur;
            WRITELN ('Besarnya uang lembur= ', Uang _ Lembur)
        END;
END.
```

2. KEPUTUSAN BERSUSUN

Untuk melaksanakan keputusan bersusun (adanya pilihan di antara beberapa alternatif), kita dapat memakai struktur bersarang atau bersusun dari **IF-THEN-ELSE**, yakni adanya **IF-THEN-ELSE** di dalam **IF-THEN-ELSE** yang lain. Berikut ini contoh sebuah struktur keputusan bersusun:

```
IF K1 THEN
    IF K2 THEN
    S1
    ELSE
    S2
ELSE
    IF K3 THEN
        S3
    ELSE
        S4
```

Diagram Alur dari struktur khusus tersebut terdapat pada gambar berikut ini :



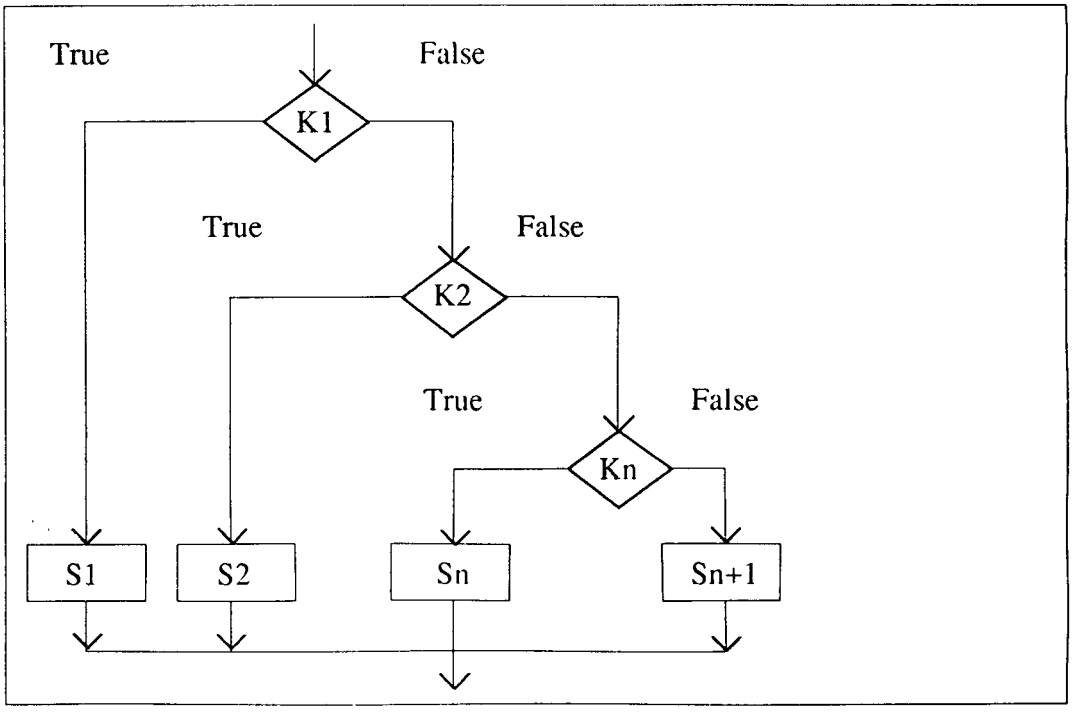
GAMBAR 3.6

Biasanya akan lebih mudah mengendalikan algoritma (atau Program), jika algoritma tersebut tidak mengandung Statemen keputusan bersusun pada jalur **THEN**, seperti pada struktur berikut ini :

```
IF K1 THEN
    S1
ELSE
    IF K2 THEN
        S2
    ELSE


        IF Kn THEN
            Sn
        ELSE
            Sn+1
```

Diagram Alur dari keputusan bersusun tersebut dapat digambarkan sebagai berikut:



GAMBAR 3-7 Struktur bersusun **IF-THEN-ELSE**

CONTOH

Berikut ini Program Pascal yang memuat Struktur **IF-THEN-ELSE** yang bersusun.

```
PROGRAM Akar _ Persamaan _ Kuadrat;
VAR
    A,B,C,D  : Real;
BEGIN
    Read (A,B,C);
    D := SQR(B) - 4 * A * C;
    IF (D < 0) THEN
        WRITELN('Akar adalah Khayal')
    ELSE
        IF (D = 0) THEN
            WRITELN('Akar Nyata Kembar')
        ELSE
            WRITELN('Akar Nyata Berlainan')
END .
```

3. TABEL KEPUTUSAN

Jika ada beberapa Kondisi yang kombinasinya akan berpengaruh pada kelanjutan pelaksanaan jalur Program, kita dapat memakai alat yang disebut Tabel Keputusan atau Decision Table, untuk menganalisisnya.

Suatu Tabel Keputusan akan merinci daftar susunan tindakan yang harus diambil oleh Pemrogram, jika terdapat Kondisi tertentu. CONTOH dari suatu Tabel Keputusan adalah sebagai berikut:

Judul		
Daftar Kondisi yang ditetapkan	Pemasukan Kondisi (True atau False)	bagian Kondisi
Daftar Tindakan yang diambil	Pemasukan tindakan (X atau kosong)	bagian tindakan

Dengan demikian, suatu Tabel terdiri dari 4 bagian, dua di sebelah kiri berisi Daftar semua Kondisi dan tindakan yang mungkin terjadi. dan dua di sebelah kanan, yang diisi dengan tanda X pada tindakan yang akan diambil untuk terjadinya Kondisi tertentu (yang tersusun vertikal di atasnya).

Sebuah Tabel Keputusan dirancang sebagai berikut :

(a) Catat seluruh Kondisi yang mungkin terjadi
(b) Catat seluruh tindakan yang mungkin diambil
(c) Tentukan sejumlah ketentuan sebagai kombinasi Kondisi yang mungkin ada (kombinasi True dan False), kecuali kombinasi yang tidak mungkin atau tidak diperlukan dalam masalah
(d) Untuk setiap Pemasukan nilai Kondisi, beri tanda X pada tindakan yang akan diambil

Pada contoh di bawah ini diberikan gambaran tentang pelaksanaan petunjuk di atas.

CONTOH 3-8.

Sebuah Tabel Keputusan untuk menghitung upah mingguan, berisi data Logika berupa tarif standar gaji perjam untuk 40 jam kerja atau kurang selama seminggu, upah untuk pekerjaan yang dikerjakan pada akhir minggu adalah dibayar lipat dua. Kemudian, nama pekerja yang bekerja pada akhir minggu, *tetapi* yang bekerja *tidak* 40 jam seminggu, dicetak namanya.

UPAH MINGGUAN

NO.	URAIAN	1	2	3	4	5
1.	Bekerja dalam 40 jam seminggu	F	F	F	T	T
2.	Bekerja kurang dari 40 jam seminggu	F	T	T	F	F
3.	Bekerja pada akhir minggu	T	F	T	F	T
1.	Hitung upah dengan tarif standar		X	X	X	X
2.	Hitung upah akhir minggu	X		X		X
3.	Cetak nama Pekerja	X		X		

Perhatikan bahwa di dalam contoh di atas,terdapat dua kombinasi Kondisi , yakni bekerja 40 jam dan bekerja kurang dari 40 jam (T T T serta T T F) yang tidak mungkin terjadi, sehingga keduanya tidak dimasukkan Tabel. Demikian pula Kondisi bahwa pekerja tidak bekerja sama sekali (F F F), juga tidak dimasukkan.

Selanjutnya, sebuah Tabel dapat disederhanakan dengan memakai simbol "diabaikan "(-). yang menunjukan Kondisi tertentu adalah tidak relevan.

Berikut ini, kita lihat Tabel Keputusan yang lalu, yang telah disederhanakan melalui pengabungan ketentuan Kondisi 2 (T F F) dan Kondisi 4 (F T F).

UPAH MINGGUAN

NO.	URAIAN	1	2	3	4
1.	Bekerja 40 jam seminggu	F	-	F	T
2.	Bekerja kurang dari 40 jam seminggu	F	-	F	F
3.	Bekerja pada akhir minggu	T	F	T	T
1.	Hitung upah dengan tarif standar		X	X	X
2.	Hitung upah akhir minggu	X		X	X
3.	Cetak Nama Pekerja	X		X	

Jadi, jika Pekerja tidak bekerja pada akhir minggu, maka proses yang sama juga diaplikasikan ,yakni menghitung upah secara standar, baik ia bekerja 40 jam ataupun kurang, tanpa menghitung upah akhir minggu.

Sesudah itu dapat dibuat Tabel Keputusan memakai bentuk Statemen **IF-THEN-ELSE.**

Variabel Logika yang dapat dipakai adalah PENUH, KURANG dan AKHMINGGU untuk menyatakan tiga Kondisi kita tersebut di atas. Tabel Keputusan di atas, dapat disusun dalam Kode-Semu Pascal, sebagai berikut:

```
BEGIN
    IF NOT AKHMINGGU THEN
        hitung upah secara standar
    ELSE
        IF PENUH THEN
            BEGIN
                hitung upah secara standar;
                hitung upah akhir minggu dan tambahkan sebagai jumlah
            END
        ELSE
                BEGIN
                    hitung upah akhir minggu;
                    cetak namanya;
                IF KURANG THEN
                    hitung upah secara standar
                    dan tambahkan ke jumlah
                END
END.
```

Perhatikan bahwa beberapa usaha perlu dikerjakan untuk mendapatkan susunan Statemen akhir yang baik, dan biasanya kita masih perlu pula mengadakan modifikasi di dalam Program yang sesungguhnya.

Dipakainya Tabel Keputusan akan memaksa Pemrogram untuk mempertimbangkan secara sistematis semua Kondisi yang mungkin terjadi, dan semua tindakan yang perlu diambil. Karenanya cara tersebut membantu dalam usaha merancang Program, di samping adanya Diagram Alur dan Kode-Semu.

3.3 GELUNG (LOOP atau PROSES BERULANG)

Suatu **Gelung** adalah pelaksanaan serangkaian Statemen secara berulang, sepanjang Kondisi tertentu masih berlaku. Jika Kondisi tersebut tak berlaku, maka Statemen terakhir di dalam Gelung akan menginstruksikan bagaimana kelanjutan proses.

Jika memasuki sebuah Gelung maka Statemen yang berada di dalamnya (disebut *body* dari Gelung) harus memungkinkan proses beralih ke Statemen awal pemasukan atau lanjutan Gelung, jika tidak terjadi demikian maka proses akan berulang terus tak berhingga. Hal ini tentunya merupakan petunjuk adanya kesalahan di dalam Pemrograman, dan Program tersebut perlu dihentikan dahulu, untuk diperbaiki.

Setiap kali body dari Gelung tersebut dilaksanakan, paling tidak satu nilai Variabel akan berubah. Oleh karena itu suatu Proses Berulang menimbulkan pengaruh yang bersifat kumulatif.

Bentuk dari Statemen Gelung **WHILE-DO** adalah :

WHILE K DO S

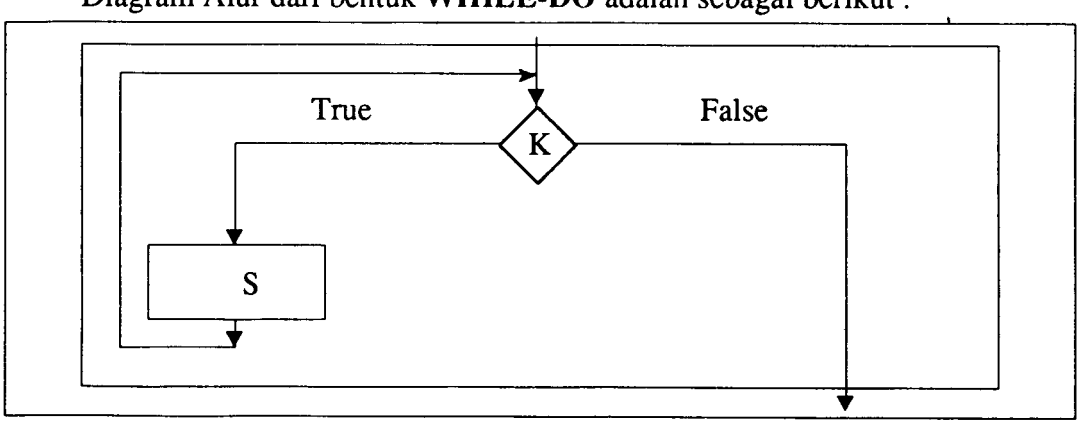
di sini K merupakan suatu Kondisi, dan S merupakan satu, atau sekelompok Statemen.

Statemen Gelung tersebut dilaksanakan sebagai berikut :

(a) Kondisi K dicoba dahulu

(b) Jika nilai K adalah *True*, S dilaksanakan,dan pengendalian Program dikembalikan ke Statemen **WHILE**, karena Kondisi tersebut akan dicoba lagi. Jika tidak (jika nilai K adalah *False*), maka akan dilanjutkan dengan Statemen sesudah S (ke luar Gelung).

Diagram Alur dari bentuk **WHILE-DO** adalah sebagai berikut :



GAMBAR 3-8. Gelung **WHILE-DO**

Memakai bentuk Statemen

WHILE K **DO**

S

merupakan cara yang menguntungkan.

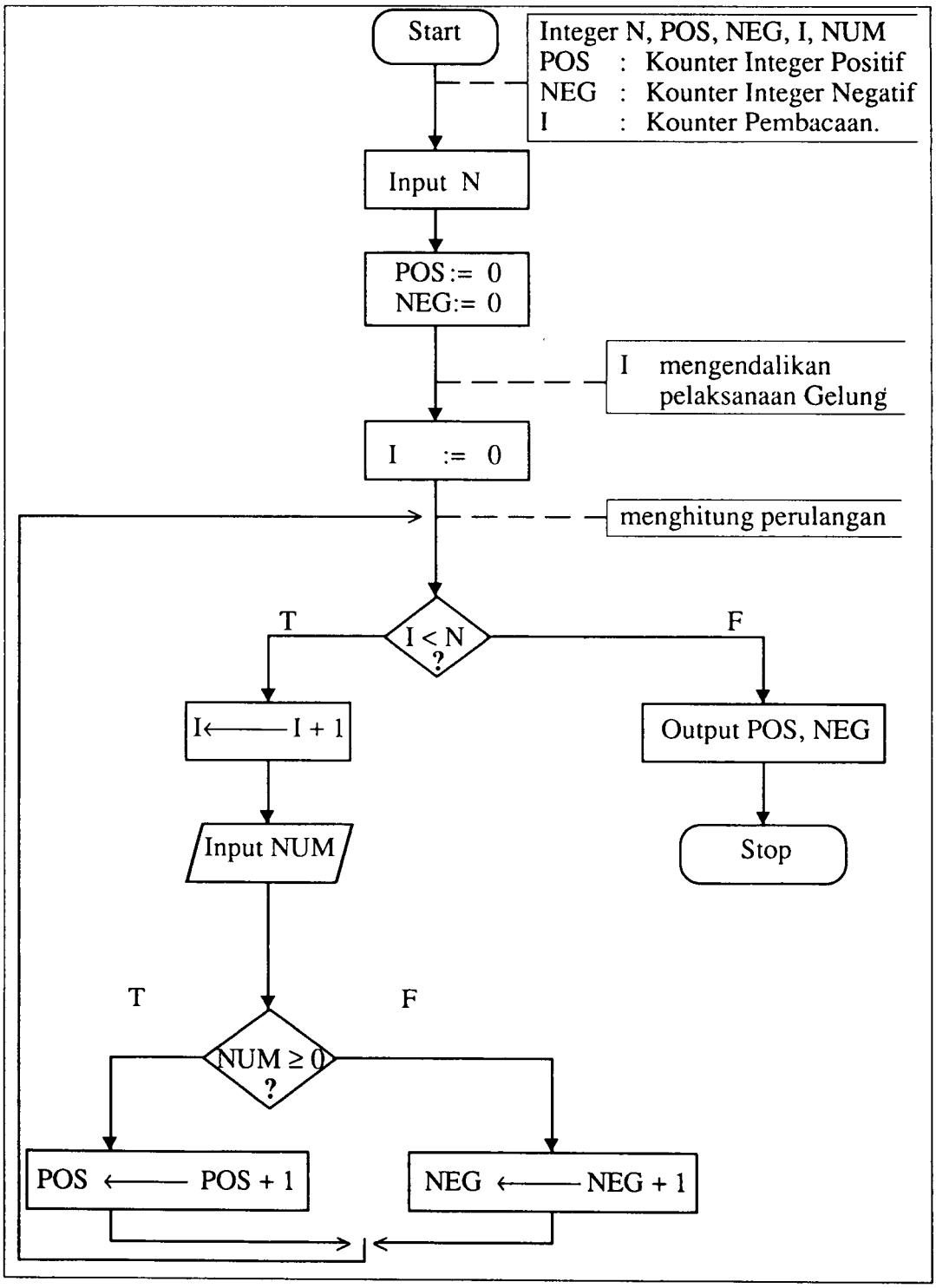
Di dalam menjawab masalah pada contoh 3-7, kita memerlukan Statemen (dalam Kode-Semu Pascal):

WHILE sudah dibaca kurang dari N integer**DO**
BEGIN
baca integer berikutnya;
tambahkan 1 pada kounter pembacaan;
tambahkan 1 pada kounter integer yang sesuai
END

Perhatikan bahwa jika K adalah *False* sesaat sebelum memasuki Statemen Gelung, maka S tidak akan dilaksanakan sama sekali. Jika K dari awal sudah *True*, dan S dilaksanakan, maka K akan menjadi **False** sesudah terulang beberapa kali.

Gambar di bawah ini merupakan jawaban terhadap Masalah di dalam CONTOH 3-7. Di sini diperlihatkan dua cara penyelesaian :

Algoritma disajikan dengan Diagram Alur (Gbr. 3 – 10)



GAMBAR 3-10

CATATAN

1. Di dalam gambar 3-10, Ulasan bertujuan untuk memberi gambaran tentang teknik yang digunakan. Teknik ini biasanya berorientasi pada bahasa Pemrograman yang akan dipakai.
2. Satu lokasi tunggal seperti NUM, dipakai untuk menampung integer yang masuk. Cara tersebut membuat memori dapat terpakai secara ekonomis, karena pada satu saat bagian proses, hanya satu integer saja yang tersedia dalam memori.

Jika Masalah Kita, mengharuskan bahwa seluruh himpunan disimpan di dalam memori, kita dapat memakai model Larik NUM(I).

Kita bisa melakukannya dengan mengganti NUM dengan NUM(I) dengan deklarasi

Integer NUM(N)

harus dimasukkan ke dalam Program tersebut.

Algoritma disajikan dalam Tata Bahasa Pemrograman (Kode-Semu)

```
{PROGRAM MENGHITUNG BanyaknyaINTEGER POSITIF DAN NEGATIF}
VAR N, POS, NEG, I, NUM : Integer;
BEGIN
        {PEMBERIAN NILAI AWAL}
        READ(N);
        POS :=   0;    {KOUNTER  INTEGER POSITIF}
        NEG :=   0;    {KOUNTER  INTEGER NEGATIF}
        I   :=   0;    {KOUNTER  INTEGER YANG MASUK}
        {PENCACAHAN}
        WHILE I<N DO
                BEGIN
                    I := I + 1;
                    READ(NUM);
                    IF NUM >= 0 THEN
                            POS   :=    POS + 1
                    ELSE
                            NEG   :=    NEG + 1
                END;
        WRITELN(POS, NEG)
END .
```

Meskipun kita sebutkan di atas, bahwa Algoritma kita merupakan sebuah Kode-Semu, namun Pembaca dapat memeriksa bahwa Ia sudah merupakan sebuah Program Pascal yang lengkap.

3.4 ALIH JALUR TANPA SYARAT

Pemakaian bentuk **if-THEN-ELSE, WHILE-DO**, dan **BEGIN-END** sebenarnya sudah cukup untuk menyusun suatu Algoritma.

Jika Program dibuat dalam susunan seperti itu, maka organisasinya boleh dikatakan akan cukup jelas, dan tersusun baik. Walaupun demikian, kadang-kadang sulit juga untuk memasukkan susunan seperti itu ke dalam suatu Algoritma tertentu. Untuk Kondisi tertentu, kita dapat memakai *sistem alih jalur tanpa syarat* (Statemen **GO TO**). Statemen tersebut memerintahkan pelaksanaan suatu Statemen tertentu tanpa memperdulikan tempatnya di dalam Program. Prosesnya berlangsung setelah perintah tersebut selesai dikerjakan.

Bentuk Statemen tersebut adalah sebagai berikut :

GO TO Label

Suatu *Label* merupakan arah simbolik dari suatu Program. Label dimasukkan ke dalam Program dengan menuliskannya di muka suatu Statemen, dan dibatasi dengan simbol khusus tertentu. Suatu Statemen hanya perlu diberi Label jika ada hubungan dengan Statemen lain, seperti Statemen **GO TO**. Pada Bahasa BASIC, Nomor Baris merupakan Label, sekaligus untuk menunjukkan urutan Statemen.

Label dapat digolongkan sebagai Identifier, karenanya penamaan Label *biasanya* mengikuti cara penamaan Identifier lain, seperti Variabel, Konstanta,Judul-Program, dan sebagainya.

Walaupun demikian, di dalam beberapa bahasa seperti FORTRAN serta BASIC, Label hanya boleh terdiri dari barisan angka saja. Sementara Turbo Pascal, memperkenankan penulisan Label, baik dalam bentuk Angka maupun bentuk Kata.

Label memberi identitas kepada suatu Statemen; jadi jangan membuat dua Statemen, di dalam sebuah Program, dengan Label yang sama.

CONTOH 3-9

Sebuah Program dapat memuat Statemen sebagai berikut :

```
BEGIN
                .
                .
        GO TO HITUNG;
                .
                .
                .
    HITUNG; NILAI := 0;
                .
                .
                .
END.
```

Sesudah pelaksanaan Statemen **GO TO**, proses akan beralih ke Statemen yang memuat instruksi dengan awalan HITUNG. Statemen yang terdapat di antara Statemen **GO TO** dan Label tersebut, tidak akan dilaksanakan.

Alternatif lain dari Statemen **GO TO**, adalah dibuatnya Statemen *exit* yang terdapat di dalam beberapa bahasa Pemrograman. Statemen tersebut banyak dipakai sebagai pengganti **GO TO.**

3.5 DARI: PERANCANGAN ALGORITMA MENGGUNAKAN KODE TULISAN, SAMPAI KE: PROGRAM

Sebuah Algoritma yang dibuat untuk memecahkan suatu masalah disusun secara bertahap. Proses pembuatan Perancangan menghasilkan Algoritma yang perlu disempurnakan terus menerus dan bentuknya sangat umum, untuk kemudian diberi kode dalam bahasa Pemrograman yang diinginkan. Teknik penyusunan Perancangan Algoritma (atau Program) dikenal dengan nama *Perancangan penyempurnaan bertingkat* atau *Perancangan dari atas ke bawah (top-down).* Tingkat-tingkat penyempurnaan yang perlu dilaksanakan tergantung dari kompleksitas algortima yang bersangkutan. Untuk menyusun sebuah Algoritma sering dipakai dua bentuk penyajian. Yang **pertama**, telah kita ketahui adalah penyajian Diagram alur, dan bentuk **kedua** memakai penyajian Kode Tulisan. **Kode-Tulisan** merupakan suatu teks dalam bahasa Indonesia (atau Inggeris) yang dimasukkan ke dalam sebuah Algoritma, berisi perintah tugas, mirip dengan bahasa mesin yang akan dilaksanakan oleh mesin komputer dengan struktur pengendalian atas dasar logika. Penyajian secara

Kode-Tulisan ini dapat dibuat mengikuti Sintaks serta Struktur suatu Bahasa Pemrograman tertentu, misalnya bahasa Pascal, dikenal pula sebagai penyajian *Bahasa-Semu,* atau *Pseudolanguage* atau*Metalanguage*, ataupun *bahasa hipotetik.*

Dengan demikian, sebuah Algoritma hasil penyempurnaan tingkat terakhir, telah melalui berbagai proses perbaikan. Akhirnya, sebuah Algoritma dapat disusun dari Statemen dasar yang telah diuraikan di dalam Bab ini, dengan penekanan lebih rinci dalam bahasa Pemrograman Pascal.

Kode-Tulisan di dalam sebuah Algoritma merupakan bagian dari Dokumentasi Program.

Berikut ini diberikan dua Contoh Masalah yangkita bahas secara bertahap, mulai dari Penulisan dalam Kode-Tulisan, sampai menjadi sebuah Program Utuh, dalam Bahasa Pascal

CONTOH 3-10. Algoritma Aritmetika

MASALAH:

Diminta untuk menentukan Pembagi Persekutuan Terbesar (PPB) dari dua integer Positif. PPB dari dua integer adalah integer terbesar yang dapat membagi kedua integer tersebut.

PEMECAHAN :

Pemecahan masalah dilakukan dengan Diagram alur, dan Kode- Tulisan.

1. *Uraian Algoritma:*

   Integer yang besar dibagi oleh yang kecil. Jika sisanya adalah 0, maka integer yang lebih kecil merupakan hasil yang diinginkan. Jika sisa tidak sama dengan nol, maka integer yang besar akan diabaikan, dan integer yang lebih kecil dinyatakan sebagai integer yang lebih besar dan sisa pembagian tadi dijadikan integer yang kecil, demikian seterusnya diulang, sampai diperoleh sisa sama dengan nol.

2. *Uji Algoritma :*

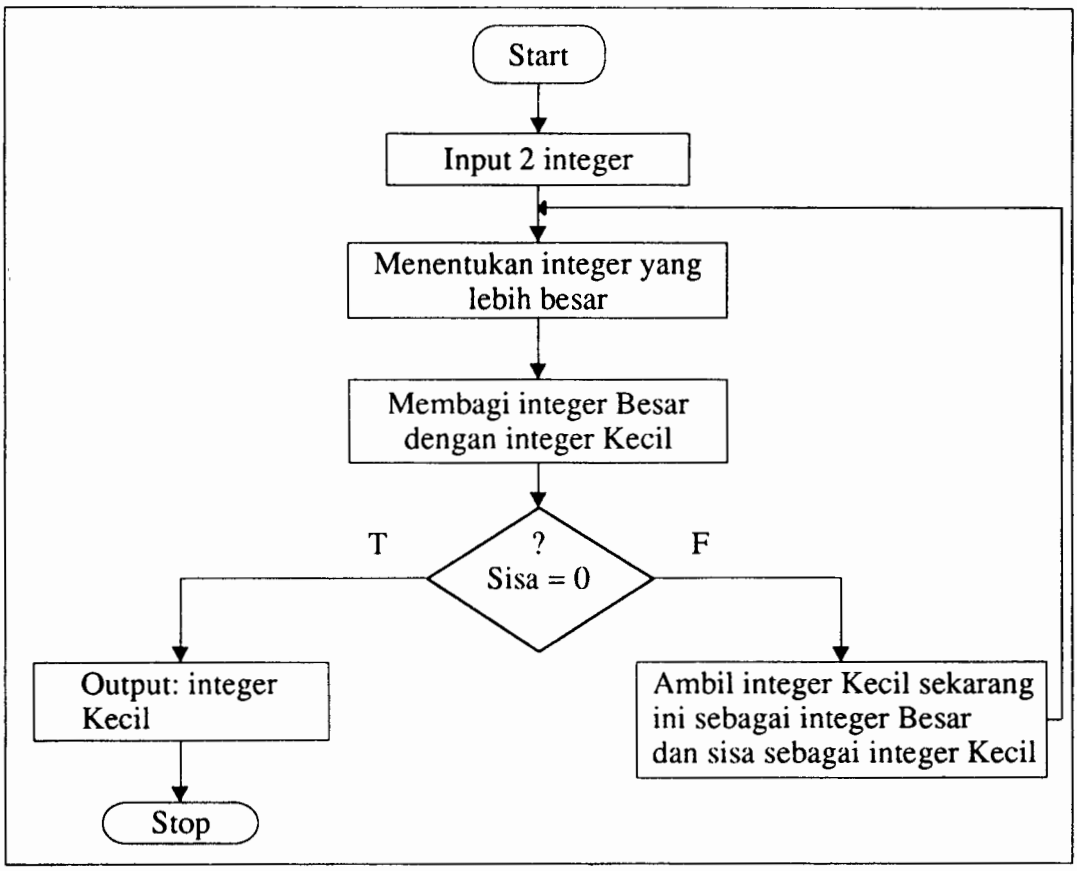
   Algoritma diuji memakai integer 12 dan 46.

   46 : 12 = 3 sisa 10

   12 : 10 = 1 sisa 2

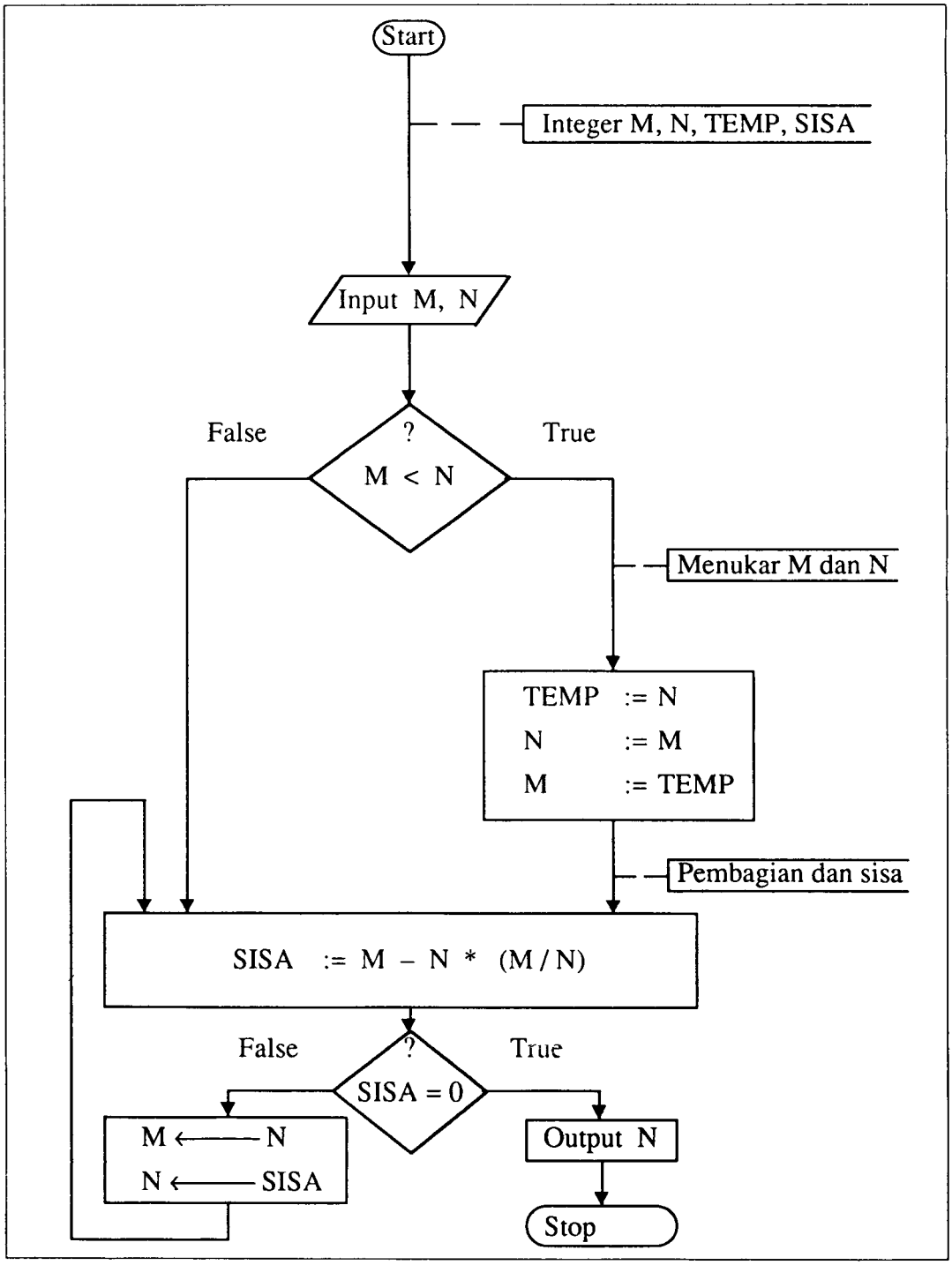
   10 : 2 = 5 sisa 0

3. *Diagram Alur awal dari Algoritma, terdapat pada gambar 3-11.*


Start
Input 2 integer
Menentukan integer yang lebih besar
Membagi integer Besar dengan integer Kecil
T
?
Sisa = 0
F
Output: integer Kecil
Ambil integer Kecil sekarang ini sebagai integer Besar dan sisa sebagai integer Kecil
Stop

GAMBAR 3-11

4. *Diagram Alur yang disempurnakan :*



GAMBAR 3 – 12

5. Algoritma dibuat dengan bahasa Pemrograman Pascal

```
PROGRAM PPB:
{ ALGORITMA ARITMETIKA UNTUK MENGHITUNG PPB DARI DUA
INTEGER POSITIF, MISALNYA M DAN N }
VAR
        M, N, TEMP, SISA : Integer;
BEGIN
        READ( M, N);
        {MENENTUKAN INTEGER YANG LEBIH BESAR}
        IF( M < N)THEN
            BEGIN
                    TEMP := N;
                    N    := M;
                    M    := TEMP
            END
        ELSE;
            {MENGHITUNG SISA}
            SISA := M - N * (M / N);
        {MENGULANG PERHITUNGAN SISA DAN PRGANTIAN NILAI
        SAMPAI SISA = 0}
        WHILE SISA <> 0 DO
            BEGIN
                    M    := N;
                    N    := SISA;
                    SISA := M - N * (M / N)
            END;
        WRITELN(N)
END.
```

Perhatikan :
Supaya Statemen **WHILE** dilaksanakan lebih dahulu, maka Variabel SISA harus diberi nilai. Dengan demikian, Statemen tersebut harus diulang terus untuk menghitung sisanya.

Apakah Algoritma tersebut dapat mudah dimengerti dalam susunan seperti di atas?

CONTOH 3-11. "Bubble Sort"

Menyortir dan mencari data merupakan kegiatan yang umum dilaksanakan oleh komputer. Terdapat banyak Algoritma untuk melaksanakan proses tersebut. Di bawah ini terdapat sebuah contoh Algoritma sortir,*Bubble Sort*. walaupun tidak begitu efisien tetapi cukup sederhana, sehingga banyak dipakai untuk menyortir sejumlah data.

Ada sejumlah integer yang akan disortir, dalam urutan menaik (dari kecil ke besar). Jadi semakin besar nilai integer tersebut, semakin belakang urutannya dalam daftar, atau semakin besar nomor urutnya.

Uraian isi Algoritma

Sebuah "bubble sort" bekerja dengan menjalankan setiap elemen, artinya membandingkannya dalam bentuk pasangan dengan elemen yang lain. Pasangan tersebut harus diganti jika bilangan-bilangannya berada pada urutan yang berlawanan dengan urutan yang diinginkan (misalnya dari kecil ke besar). Satu bilangan umumnya tidak dapat "dijalankan" meliwati seluruh bilangan yang ada. Bilangan dijalankan berulang kali, hingga pada langkah terakhir tidak perlu ada perubahan lagi. Kondisi tersebut menunjukkan bahwa bilangan sudah menempati urutan yang semestinya. Bilangan "bubble up" dari "bawah" dilakukan untuk mengurutkan tempatnya di dalam daftar himpunan. Aplikasi dari Algoritma untuk sebuah contoh himpunan bilangan terlihat di bawah ini :

i	DAFTAR TAK TERURUT	SESUDAH ITERASI 1	SESUDAH ITERASI 2	SESUDAH ITERASI 3	SESUDAH ITERASI 4
7	13	17 ←	17	17	17
6	10	13	13	13	13
5	-10	10	10	10	10
4	17	-10	2 ←	2	2
3	-15	2 ←	-10	-3 ←	-3
2	2	-15	-3 ←	-10	-10
1	-3	-3	-15	-15	-15

Tanda ⟵ dipakai untuk menunjukkan "bubbling". Proses dari langkah pertama dapat dilihat dalam daftar berikut ini. Kolom berurutan menunjukkan langkah proses perbandingan.

Tanda ⟵ menandakan terjadi pergantian bilangan

Tanda ⟵ menandakan tidak ada perubahan.

INITIAL	I	II	III	IV	V	VI
13	13	13	13	13	13	17
10	10	10	10	10	17	13
-10	-10	-10	-10	17	10	10
17	17	17	17	-10	-10	-10
-15	-15	2	2	2	2	2
2	2	-15	-15	-15	-15	-15
-3	-3	-3	-3	-3	-3	-3

Terlihat bahwa bilangan terakhir tergeser ke atas setelah melalui sejumlah langkah dan menduduki posisinya yang terakhir. Suatu bubble sort yang berurutan ke atas merupakan sebuah bubble sort yang dapat menghemat waktu karena kita tidak bergerak di atas angka-angka yang sudah dijalankan sebelumnya; lagi pula jika angka terakhir dijalankan pada posisi kedua dari bawah, maka sort tersebut sudah selesai dan tidak perlu melakukan langkah tambahan tanpa perubahan.

Kode-Semu; Deskripsi awal

```
*BUBBLE SORT
    BEGIN
            Input Larik (array);
            WHILE diharapkan ada pertukaran DO
                lakukan pass berikutnya pada larik (array);
            Output (sorted) array
    END •
```

Kode-Semu : Perbaikan Pertama

```
*BUBBLE SORT  DALAM URUTAN ASCENDING*
    BEGIN
        Input array;
        beri tanda LIMIT; *LIMIT-INDEX DARI ITEM TERAKHIR
                    UNTUK DIBANDINGKAN*
    *KERJAKAN PASS BERTURUTAN
        WHILE diharapkan adanya pertukaran DO
            BEGIN
                mulai dari dasar;
    *KERJAKAN SUATU PASS TUNGGAL*
```

```
                WHILE lebih rendah dari LIMIT dan limit = 1 DO
                    BEGIN
                        bandingkan sepasang-sepasang dan
                        pertukarkan bila perlu;
                        pindah ke pasangan berikutnya
                    END
                beri tanda LIMIT baru
                END
            Output (sorted) array
        END
```

Kode-Semu : Perbaikan Kedua

```
*BUBBLE SORT DALAM URUTAN ASCENDING*
    BEGIN
            Integer A(N), LIMIT, LAST, TEMP, I;
    *INPUT ARRAY
            Input N;
            I     :=    I+1
            WHILE I ≤ N DO
                BEGIN
                        Input A(I);
                        I     :=    I + 1
                END;
    *TANDAI LIMIT-INDEX DARI ITEM TERAKHIR
            LIMIT     :=    N;
    *KERJAKAN PASS BERTURUTAN*
    *JIKA LIMIT = 0, TAK ADA PERTUKARAN DALAM*
    *PASS TERAKHIR;JIKA LIMIT = 1, BILANGAN ADALAH DALAM*
    *URUTAN*
            WHILE LIMIT > 1 DO
                BEGIN
                        I     :=    1;
    *MENGAWALI LAST-INDEX DARI ITEM TERAKHIR PINDAH*
    *DALAM PASS INI*
                LAST      :=    0;
    *KERJAKAN SUATU PASS TUNGGAL*
            WHILE I < LIMIT DO
                BEGIN
    *PERTUKARKAN PASANGAN BILA PERLU*
            IF A(I) > A(I+1) THEN
```

```
            BEGIN
                TEMP  :=  A(I);
                A(I)  :=  A(I+1);
                A(I+1) :=  TEMP;
                LAST  :=  I
            END
        ELSE;
    *PINDAH KE PASANGAN BERIKUTNYA*
            I     :=    I + 1;
        END;
    *TANDAI LIMIT BARU*
            LIMIT     :=    LAST
        END;
    *OUTPUT SORTED ARRAY*
        I     :=    1;
        WHILE I ≤ N DO
            BEGIN
                Output A(I);
                I     :=    I + 1
            END
END.
```

Perhatikan :

1. Algoritma bekerja dengan tepat untuk kondisi N = 0 atau N=1;
2. Memakai variabel LAST dan LIMIT;
3. Studi modifikasi terhadap penyempurnaan-penyempurnaan Algoritma dari bentuknya yang semula;
4. Aplikasikan Algoritma tersebut dengan bilangan, kemudian ikuti :
   a. langkah-langkah berturut-turut terhadap himpunan tersebut;
   b. nilai Variabel LAST dan LIMIT.

3.6 STRUKTUR PENGENDALIAN TAMBAHAN

Bentuk Statemen **BEGIN-END,IF-THEN-ELSE,** dan **WHILE-DO** sesungguhnya sudah cukup untuk menyusun sebuah Algoritma. Tetapi, rasanya masih dibutuhkan beberapa bentuk lainnya.

Dua di antaranya adalah mekanisme Gelung, yakni Gelung **REPEAT-UNTIL** dan Gelung berindeks, sedangkan yang ketiga adalah Statemen untuk aneka pilihan, yakni **CASE**, ataupun dapat juga dipakai Statemen exit.

1. BENTUK GELUNG ALTERNATIF

Ada dua bentuk Gelung Alternatif yang kadang-kadang juga dipakai, yakni Gelung **REPEAT-UNTIL** dan Gelung berindeks atau Gelung **FOR-DO**. Di dalam Gelung **REPEAT-UNTIL** Kondisi Gelung di tes sesudah pelaksanaan body Gelung, sedangkan pada Gelung WHILE-D0 dilakukan sebelumnya.

Bentuk Gelung berindeks secara otomatis mempertahankan Variabel yang menentukan jumlah gerakan langkah.

A. GELUNG **REPEAT-UNTIL**

Di dalam situasi Pemrograman tertentu, kerap kali kita perlu melaksanakan sekelompok Statemen secara berulang, sambil melihat apakah timbul suatu Kondisi tertentu. Jika hal ini terjadi, diinginkan agar pelaksanaan segera dialihkan ke Statemen berikutnya, yakni kita keluar dari Gelung.

Untuk mendapatkan bentuk Gelung seperti itu, kita dapat menggunakan Struktur **REPEAT-UNTIL**, yang bentuk umumnya adalah :

REPEAT S UNTIL C

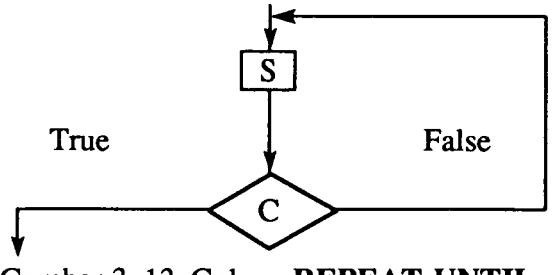
di sini S merupakan sebuah atau sekelompok Statemen, sedangkan C merupakan suatu Kondisi.

Gelung tersebut dilaksanakan sebagai berikut :

1. Statemen S dilaksanakan
2. Kondisi C diuji
3. Jika C adalah False, pengendalian Program dikembalikan, kepada Statemen **REPEAT**, sehingga S dilaksanakan lagi.
   Jika C adalah True, maka Statemen sesudah Gelung akan dilaksanakan.

Agar diperhatikan, bahwa Statemen / kelompok Statemen S sekurang-kurang-nya dilaksanakan satu kali. Jika S dilaksanakan lagi, maka pelaksanaan ulang tersebut harus mengubah nilai dari Kondisi C menjadi True.

Diagram Alur dari bentuk **REPEAT-UNTIL** terdapat dalam gambar 3-13.



Gambar 3–13 Gelung **REPEAT-UNTIL**

Dianjurkan untuk menyusun bentuk tersebut sebagai berikut :

REPEAT

S

UNTIL C

Bentuk Gelung **REPEAT-UNTIL** dapat dipakai pula sebagai pengganti bentuk **WHILE-DO**, jika bentuk tersebut ternyata kurang efektif.

Contoh 3-13.

Untuk masalah pada contoh 3-10, kita juga bisa memakai Gelung **REPEAT-UNTIL.**
Bagian Algoritma untuk menentukan integer terbesar, adalah menjadi :

```
REPEAT
        BEGIN
            SISA :=   M - N * (M / N);
            M    :=   N;
            N    :=   SISA
        END
UNTIL SISA = 0;
WRITELN (M);
```

Perhatikan : Instruksi kepada SISA tidak perlu disebut ulang.

Contoh 3-13

Jika sejumlah satuan Data di dalam Program harus di proses, sebaiknya Komputer dapat menghitung banyaknya Data tersebut. Di sini kita perlu memberi tanda, berupa suatu Data semu. Data Semu tersebut ditaruh sebagai Data terakhir pada Himpunan Data. Nilai Data Semu ini benar-benar harus diluar daerah nilai Data yang sebenarnya. Data Semu itu biasa dikenal sebagai **Panji** atau **Bendera** atau **Flag.**

Dengan demikian didalam contoh 3-11, jika kita secara pasti mengetahui bahwa tidak ada anggota himpunan berupa Integer 0, maka Data 0 dapat dipakai sebagai Panji.

```
*RANGKAIAN INPUT ALTERNATIF UNTUK BUBBLE SORT
I     :=    0;
REPEAT
          BEGIN
               I     :=    I + 1;
               Input A(I)
          END
UNTIL A(1) = 0;
N     :=    I - 1;
```

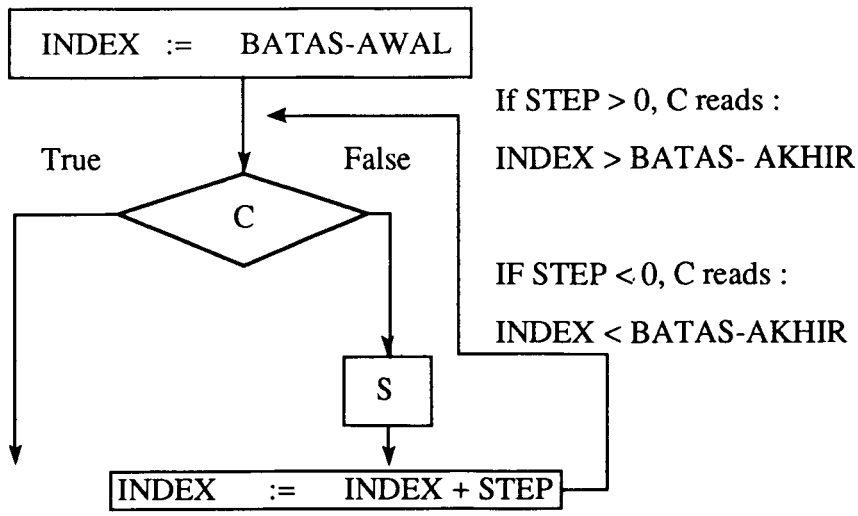
Perhatikan bahwa beberapa Bahasa Pemrograman memiliki mekanisme yang jelas untuk menguji akhir dari suatu Berkas Input, yakni adanya Reserved-word **EOF** (END OF FILE).

B. GELUNG BERINDEKS

Gelung berindeks (juga disebut Gelung berhitung atau Gelung **FOR-DO**), dapat secara otomatis mengendalikan Gelung dengan menentukan jumlah pengulangan. Dalam Bahasa BASIC, Struktur ini terkenal sebagai Struktur **FOR-NEXT.** Bentuk umum dari Statemen Gelung berindeks adalah

FOR INDEX := BATAS-AWAL UNTIL BATAS-AKHIR BY **STEP DO**

Diagram Alur dari bentuk Statemen tersebut terdapat di dalam Gambar 3-14, yang akan memberi penjelasan tentang pelaksanaannya.



Gambar 3.14 Gelung berindeks

Empat Variabel yang dipakai dalam bentuk tersebut semuanya harus memiliki nilai integer. Perhatikan bahwa Gelung tersebut "melakukan pekerjaan lebih banyak" dibandingkan dua bentuk lainnya, karena : diawali dan mempertahankan Gelung berindeks, dan sekaligus mengecek penyelesaiannya. Dengan demikian, bentuknya ringkas dan dapat diaplikasikan menurut kebutuhan.

Contoh 3-14.

Jika kita ingin mendapatkan jumlah dari 50 integer positif yang pertama, kita dapat memakai Gelung berindeks berikut ini :

```
SUM :=   0;
I    :=   1;
WHILE I < 50 DO
    BEGIN
            SUM :=   SUM + I;
            I    :=   I + 1
    END;
```

2. ANEKA PILIHAN (CABANG GANDA)

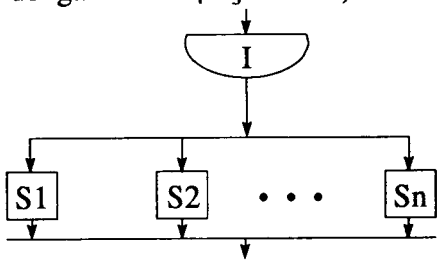
Jika ingin membuat keputusan yang menghasilkan beberapa jawaban, agar memakai bentuk **CASE.** Bentuk Statemen **CASE** adalah sebagai berikut :

```
CASE I
    BEGIN
        S1;
        S2;
        •
        •
        •
        Sn
    END
```

di sini I merupakan suatu Variabel integer yang nilainya adalah 1hingga N > 1, dan S1, ... , Sn merupakan Statemen.

Di dalam Gambar 3-15 terdapat Diagram Alur yang menjelaskan pelaksanaan bentuk Statemen tersebut (memakai kotak yang bukan standard untuk memper-

lihatkan hasil tes dengan beberapa jawaban).



Gambar 3.15 konstruksi **CASE** 12

Sebuah statemen **Case** dilaksanakan sebagai berikut :

(a) Suatu nilai dari Variabel integer kita tetapkan.
(b) Jika nilai tersebut adalah J, maka statemen Sj akan dilaksanakan. Selanjutnya dilaksanakan statemen **Case** yang bersangkutan (yakni statemen sesudah Sn)

Pemakaian statemen Case tersebut dapat menggantikan bentuk if then-else yang kadang-kadang menimbulkan cabang keputusan yang agak rumit susunannya.

Contoh 3.15

Ada penagihan terhadap empat langganan yang harus dijumlahkan. Sebuah data tagihan terdiri dari kode langganan (dengan nilai 1sampai dengan 4) dan jumlah tagihannya. Akhir dari berkas atau file input ditandai dengan langganan kode 0. Berikut ini Algoritmanya dalam bentuk Kode-Semu.

```
*MENCARI TOTAL ACCOUNTS RECEIVABLE*
begin
        Integer CODE;
        Real DUE, TOTAL (4);
        for K ← 1 until 4 by 1 do *KEADAAN AWAL*
            TOTAL (k) ← 0;
*KLASIFIKASI DAN MENAMBAHKAN NILAI DUE (TAGIHAN)*
        Input CODE, DUE
        while CODE <> 0 do
            begin
                case CODE
                    begin
                        TOTAL (1) ← TOTAL(1) +DUE;
```

```
                    TOTAL (2) ← TOTAL(2) +DUE;
                    TOTAL (3) ← TOTAL(3) + DUE;
                    TOTAL (4) ← TOTAL(4) + DUE
                end;
            Input CODE,DUE
        end;
    for K ← 1 until 4 by 1 do
        Output TOTAL (K)
end •
```

Perhatikan :

1. Perbedaan antara statemen **case** dan statemen majemuk.
2. Anda harus yakin bahwa algoritma tersebut akan bekerja secara tepat jika sudah tidak ada data input lagi.
3. Buatlah algoritma untuk masalah yang sama, menggunakan bentuk **if-then-else,** kemudian dibandingkan.

3. EXIT

Jika timbul kondisi tertentu, kadang-kadang perlu untuk keluar (**exit**) dari gelung sebelum selesai. Hal tersebut dapat dilakukan dengan statemen go to. Cara yang lebih baik untuk keluar dari sebuah gelung adalah memakai suatu statemen exit.

Bentuk dari sebuah statemen **exit** adalah :

exit label

di sini label memberi ciri kepada statemen pertama sebagai sumber untuk melaksanakan statemen **exit** yang bersangkutan. Sesudah dilaksanakan, proses beralih kepada statemen berikutnya.

Contoh 3–16.

Sebagian Kode - Semu berikut ini, dapat dipakai untuk menggantikan sebagian dari Kode-Semu dalam contoh 3-15.

```
*KLASIFIKASI DAN MENAMBAHKAN NILAI DUE (TAGIHAN)*
KELAS : repeat
            begin
                    Input CODE, DUE;
                    if CODE = 0 then
                        exit KELAS
                    else;
                    case CODE
                        begin
                            •
                            •
                            •
                        end
            end
        until CODE = 0
```

Perhatikan bahwa sebenarnya kita dapat menuliskan **until** untuk seterusnya, karena pelaksanaan gelung **repeat-** selalu akan diselesaikan melalui statemen **exit** tersebut.

Statemen **exit** sering dipakai untuk mengecek kondisi khusus tertentu, termasuk kesalahan terhadap data yang diproses oleh gelung tersebut, contoh penting berikut ini menggunakan bentuk **exit.**

Contoh 3-17. Pencarian binar (binary search)

Mencari sambil menyortir (mengurutkan) merupakan kegiatan yang biasa dikerjakan oleh komputer. Kegiatannya meliputi kegiatan meneliti daftar item, untuk mencari item yang dibutuhkan. Tujuan dari pencarian tersebut sesungguhnya adalah untuk mendapatkan data yang tersimpan bersama item yang memberi ciri kepadanya (disebut key).

Pencarian dapat dilakukan dengan sistem *cari sekuensial (sequntial search),* yakni mulai dari item pertama dalam daftar, dibandingkan dengan item yang sedang dicari, hingga ditemukan atau hingga mencapai akhir daftar.

Pencarian secara sekuensial tersebut berlangsung lambat, suatu pencarian yang baik berlangsung rata-rata *N/2* jika daftar berisi *N* item.

Cara yang lebih cepat ialah dengan *Pencarian binar* atau *binary search,* yakni satu di antara sekian banyak algoritma yang dapat dipakai untuk mencari item di dalam daftar yang sudah tersusun itemnya.

Suatu *binary search* mirip dengan cara mencari kata di dalam kamus. Kita berasumsi bahwa daftar telah terurut dari atas ke bawah mulai nilai kecil ke nilai besar. Pertama-tama bagian tengah dari daftar tersebut diteliti, jika item yang dicari itu lebih kecil dari item di bagian atas daftar, maka pencarian dilakukan dari bagian tengah kebawah.

Data yang kita cari dapat ditemukan atau tidak, pencarian dengan sistem binar membutuhkan perbandingan rata-rata $\log_2 N$ untuk menemukan item di dalam daftar yang berisi N item. Proses ini akan lebih cepat dibandingkan dengan sistem sekuensial. Contoh berikut ini merupakan Kode-Semu algoritma tersebut.

```
*PENCARIAN BINAR
*Sebuah daftar KEY (I) terdiri N item; kita akan mencari item yang
*disebut GIVEN; posisi INDEX di dalam daftar harus ditentukan.
*Jika item tidak ditemukan, INDEX akan menjadi 0*
begin
    Integer KEY(N), GIVEN, INDEX, LOW, HIGH;
    Input N, GIVEN;
    For I ← 1 until N by 1 do
        Input KEY (I);
    LOW  ← 1; *posisi terendah dalam sub daftar sedang dicari*
    HIGH ← N; *posisi tertinggi dalam sub daftar sedang dicari*
*posisi INDEX dari item GIVEN ditentukan
SRC : while LOW =< HIGH do
    begin
        INDEX ← (LOW + HIGH) / 2; *pembagian integer*
        if GIVEN < KEY (INDEX) then
            HIGH ← INDEX - 1 *melihat separuh pertama*
        else
            if GIVEN > KEY(INDEX) then
                LOW ← INDEX + 1 *melihat separuh kedua*
            else
                exit SRC   *ditemukan*
    end;
    if LOW > HIGH then          *tidak diketemukan*
        INDEX ← 0
    else
end.
```

Perhatikan bunyi statemen keputusan terakhir tersebut.
Di bawah ini terdapat algoritma yang bersangkutan

TABEL 3-2. Langkah-langkah dalam binary search : dari 9 bilangan (N=9), untuk mencari bilangan 300 (GIVEN = 300).

KEY (I).

SEBELUM PERBANDINGAN I	SEBELUM PERBANDINGAN II	SEBELUM PERBANDINGAN III
5 ← LOW	5	5
21	21	21
73	73	73
124	124	124
INDEX → 241	241	241
300	300	→ 300
450	→ 450	450
701	701	701
715 ← HIGH	715	715
LOW = 1 HIGH = 9 INDEX = $\left[\frac{1+9}{2}\right] = 5$ ↑ Pembagian integer	LOW = 6 HIGH = 9 INDEX = $\left[\frac{6+9}{2}\right] = 7$	LOW = 6 HIGH = 6 INDEX = $\left[\frac{6+6}{2}\right] = 6$

3.7 KESIMPULAN, JENIS STATEMEN DI DALAM PEMROGRAMAN

Tujuh Statemen berikut ini diperlukan dalam bahasa pemrograman yang serba guna. Tiga di antaranya adalah bentuk pengendalian pelaksanaan program, yakni :

(1) **begin–end;**

(2) **if–then–else** (serta **if-then**);

(3) **While–do.**

Sebuah algortima yang tersusun sederhana dengan bentuk pengendalian seperti itu semata-mata disebabkan karena sistem pengerjaan adalah satu-masuk-satu-keluar. Diagram Alurnya hanyamemiliki satu jalur masuk dan satu jalur keluar.

Berhubung setiap teks ditulis dan dibaca dari atas ke bawah maka pengorganisasian arus program harus mengikuti garis umum tersebut. Beberapa di antara bentuk

tersebut jelas dipakai di dalam bahasa pemrograman tertentu, bentuk lainnya bisa disusun dari statemen yang tersedia.

Sebagai tambahan untuk statemen pengendalian arus program, berikut ini terdapat statemen penanganan Data (seperti diuraikan di alam Bab 2) :

(4) **go to;**
(5) **input/output;**
(6) **penugasan;**
(7) **deklarasi** (tidak dapat dilaksanakan).

Untuk menjelaskan logika program dan arti dari data yang bersangkutan, dianjurkan agar kita memakai ulasan (remark) seperlunya.

Untuk memudahkan Pemrograman dalam tugasnya, ada banyak statemen tambahan di dalam bahasa pemrograman tingkat tinggi.

Bentuk statemen tambahan tersebut adalah :

(1) gelung **repeat-until;**
(2) gelung **index;**
(3) statemen **case;**
(4) statemen **exit.**

Suatu fasilitas penting di dalam bahasa pemrograman serbaguna ialah pembuatan subprogram yang akan di bahas di dalam Bab 4 berikut.

Setiap bahasa memiliki ketentuan tersendiri untuk menyebut variabel, dan menetapkan label.