4 SUB PROGRAM DAN PEMBUATAN PROGRAM DENGAN SISTEM MODUL

4.1 PENDAHULUAN

Sebuah Algoritma yang agak kompleks danProgram yang tersusun dari Algoritma tersebut dapat dianggap sebagai suatu sistem. Untuk merancang suatu sistem, kita harus dapat memberi identitas kepada struktur hirarkhinya, kemudian membangun elemen Modul yang mengadakan interaksi. Perangkat tersebut terdiri dari subprogram yakni : ***serangkaian Statemen yang diberi nama tertentu, untuk melaksanakan tugas tertentu, dan dapat "dipanggil" dengan nama tersebut.***

Sebuah Program yang mengandung Subprogram, terdiri atas, Program Utama yang dilaksanakan lebih dahulu, dan Sub Program yang diminta oleh Program Utama, atau Subprogram lainnya. Program Utama bersama Subprogram disebut *Modul* atau Unit Program.

Suatu Subprogram terdiri dari dua bagian pokok, yaitu : fungsi dan subrutin (yang dapat dimanfaatkan berulang-ulang), dalam Bahasa Pascal biasanya disebut **Procedure.**

Ketika Subprogram "dipanggil," terjadi komunikasi antara dua Modul, melalui beberapa data item tertentu. Komunikasi tersebut dapat dilaksanakan dengan me-

mindahkan data secara nyata (disebut **Parameter passing**), atau pemindahan melalui lokasi memori yang memuat data bersangkutan. Cara berkomunikasi yang kedua ini, terutama terdapat dalam Bahasa Pemrograman "berkotak" atau Terstruktur, seperti ALGOL ataupun PL/I. Di sini data tertentu dianggap bersifat global terhadap subprogram, dan oleh karenanya dapat dimasukkan ke dalam Unit-Unit Program lainnya.

Jika suatu Bahasa Pemrograman memungkinkan adanya Subprogram, maka hal tersebut dapat menyederhanakan pembuatan Algoritma yang dilaksanakan dalam Bahasa tersebut.

Pembuatan desain Program yang berisi identitas tugas terinci, dan dapat memisah atas Modul disebut *Pemrograman Modular.*

4.2 PEMANGGILAN SUBPROGRAM

Seperti telah dikemukakan di atas Sub Program dapat merupakan fungsi ataupun subrutin. Ketika Program dilaksanakan, sebuah fungsi memberikan kepada Unit Program yang memintanya, berupa suatu nilai tunggal sebagai nilai dari fungsi tersebut.

Subrutin merupakan bentuk yang paling umum dari subprogram yang dapat dipakai untuk berkomunikasi dengan Program yang memanggil sejumlah nilai dari Program Unit yang bersangkutan. Walaupun sebuah fungsi dapat dipakai untuk mengembalikan sejumlah nilai dengan cara yang sama seperti yang dipakai subrutin, dianjurkan untuk tidak melakukannya. Subprogram yag terdapat dalam Program Unit memiliki seluruh sifat dari Algoritma, kecuali outputnya, karena yang akan melaksanakan proses tersebut adalah sebuah Modul lain di dalam Program yang sama.

Subprogram dapat dikompilasi tersendiri untuk pengetesan fungsinya. Sesudah pemanggilan dan pelaksanaan subprogram, pengendalian kembali ke tempat pemanggilan di dalam Unit Program yang memanggilnya.

4.3 SUBRUTIN

Sebuah subrutin merupakan bentuk yang lebih umum dari dua jenis subprogram tersebut di atas. Sebuah subrutin merupakan suatu Program yang secara nyata mengembalikan sejumlah nilai (tanpa batas) kepada Program Unit yang memanggilnya. Pemanggilan subrutin disebut *subroutine call.* Bentuk dari sebuah Statemen subroutine call adalah sebagai berikut :

CALL nama-subroutine (daftar Parameter)

Nama subrutin diberikan seperti memberi nama sebuah variabel. Nama tersebut memberi identifikasi kepada subrutin di antara sejumlah subprogram yang berada di dalam Program tertentu.

Parameter (juga disebut argumen) merupakan data item yang digunakan oleh sebuah subrutin untuk berkomunikasi dengan Program Unit yang memanggilnya. Parameter ditetapkan terlebih dahulu untuk sebuah subrutin dengan jumlah dan jenisnya masing-masing. Parameter dapat menghubungkan nilai dengan subrutin (kadang disebut input Parameter) atau dari subrutin dengan Program Unit yang memanggilnya, atau berfungsi sebagai keduanya dengan memberi nilai kepada subrutin apabila selanjutnya membutuhkan nilai dari hasil pelaksanaan Program (Parameter input/output). Parameter dapat terdiri dari nilai tunggal atau himpunan nilai.

Contoh 4.1

Untuk memanggil suatu subrutin SWAP yang dimaksudkan untuk menukar nilai-nilai dari 2 integer I dan J dapat memakai Statemen berikut ini :

Call SWAP(I,J)

Kedua Parameter tersebut adalah dari jenis input/output. Salah satu tujuan dari subprogram adalah berfungsi serba guna di dalam Program. Dengan demikian, beberapa "nama panggilan" yang diberikan kepada subprogram dapat dimasukkan ke dalam berbagai Modul Program. Selama pemanggilan sedang berlangsung, nilai-nilai lain dari Parameter yang nyata dimasukkan ke dalam daftar. Supaya pemanggilan tersebut bermanfaat, maka subrutin yang dipanggil harus ditentukan isinya dan dimasukkan ke dalam Program.

Sebuah subrutin, terdiri dari *Judul* subrutin , dengan bentuk sebagai berikut :

SUBROUTINE nama-subrutin(daftar Parameter)

dan *Badan* atau Body dari subrutin yang bersangutan, yakni Statemen yang menentukan subrutin tersebut. Bentuk tulisan dari sebuah subrutin body mirip dengan Program utamanya. Sesudah pelaksanaan Statemen terakhir dari sebuah subrutin, pengendalian Program akan kembali kepada Program Unit yang memanggilnya, khususnya kepada teks Statemen yang berada di bawah Statemen "call" tersebut.

Judul subrutin berisi Parameter formal dari subrutin yang bersangkutan, dan nama variabel dan himpunan dari data yang berkomunikasi dengan subrutin dan dengan Program Unit yang memanggilnya. Nama tersebut ditulis secara terpisah, sepanjang argumen formal dan yang aktual sesuai/cocok. Parameter formal menetapkan tempat untuk argumen yang aktual. Hubungan antara Parameter formal dan aktual terjadi di saat pemanggilan subrutin yang bersangkutan. Berbagai bahasa Pemrograman memakai nama yang berlainan untuk Parameter tersebut diatas. Nama yang sering dipakai terdapat di dalam Tabel 4-1 di bawah ini.

Tabel 4.1 Istilah yang dipakai untuk lintasan Parameter.

Di dalam Modul pemanggil		Di dalam Modul yang dipanggil	Nama tersebut dalam Bahasa
Parameter nyata	⟷	Parameter formal	ALGOL, PASCAL
argumen nyata	⟷	argumen semu	FORTRAN
argumen	⟷	Parameter	PL/I

Sebuah subrutin dapat memakai sejumlah variabel logikal atau himpunan, bersama dengan nilai Parameter yang bersangkutan yang terdapat di dalamnya. Eksistensinya tidak diketahui oleh Unit-Unit Program lainnya. Pemrogram dapat bebas memilih nama resmi bagi nama variabel atau label di dalam sebuah Unit Program. Jika ada nama yang kebetulan sama, akan diperlakukan sebagai item yang berbeda. Parameter formal selalu berisi nama-nama variabel dan himpunan sedangkan Parameter yang nyata dapat berupa ekspresi-ekspresi (terutama konstanta, variabel dan elemen himpunan). Antara fungsi-fungsi aljabar dan subprogram-subprogram di dalam Pemrograman terdapat suatu analogi, karena fungsi aljabar ditulis dengan tanda-tanda nyata untuk nilai-nilai yang aktual.

Misalnya untuk mendapatkan hipotenusa (sisi miring) dari sebuah segitiga siku-siku dengan sisi x dan y kita dapat memakai rumus :

$$z = \sqrt{(x^2 + y^2)}$$

Setelah penggantian nilai x dengan nilai yang nyata (misal x=4, y=3), maka jawaban adalah z=5. Anggota dari Parameter formal dan yang aktual berada di dalam posisi yang sama.

Contoh 4.2

Studi tentang penamaan Parameter terdapat di dalam contoh berikut :

Program Unit yang memanggil

```
BEGIN
    Integer I,J(100);
    Real X,Y,Z(5);
        •
        •
    Call WORK(X,Z,J(I);
        •
        •
    Call WORK(Z(3),Z,5);
        •
        •
END.
```

Subprogram yang dipanggil:

```
Subroutine WORK(B,D,N)
BEGIN
    Integer K(100);M,N;
    Real A,B,C,D(50);
        •
        •
END.
```

Hubungan antara Parameter aktual dan formal yang terjadi pada setiap "call" adalah :

PARAMETER NO.	ACTUAL PARAMETER KETIKA CALL PERTAMA	ACTUAL PARAMETER KETIKA CALL KEDUA	FORMAL PARAMETER	TIPE PARAMETER
1	X	Z(3)	B	nilai real
2	Z	Z	D	array real 50 elemen
3	J(I)	5	N	Nilai integer

Hal yang perlu diketahui oleh penulis Program Unit yang memanggil atau yang dipanggil ialah ujud ganda di antara kedua kondisi tersebut, yakni arti dari setiap Parameter yang ada didaftar, posisi dari Parameter tersebut dan sifat-sifatnya. Sebuah subrutin dipanggil dengan cara sebagai berikut :

(1) Parameter dijalankan

(2) Subrutin dilaksanakan seolah-olah merupakan sebuah Program tersendiri. Pa-

rameter-Parameter output akan diberi nilai; Parameter input/output dapat dirobah nilainya.

(3) Pengendalian Program akan kembali kepada Statemen di bawah teks subrutin.

Beberapa bahasa Pemrograman mempunyai Statemen khusus untuk mengembalikan proses dari sebuah subrutin kepada Program utamanya (RETURN dalam FORTRAN, atau EXIT PROGRAM dalam COBOL). Di dalam bahasa ALGOL subprogram terdiri dari Statemen campuran (seperti di dalam Program Utamanya).

Contoh di bawah ini memberi gambaran tentang pelaksanan sebuah subrutin yang sederhana.

Contoh 4.3

PROBLEMA

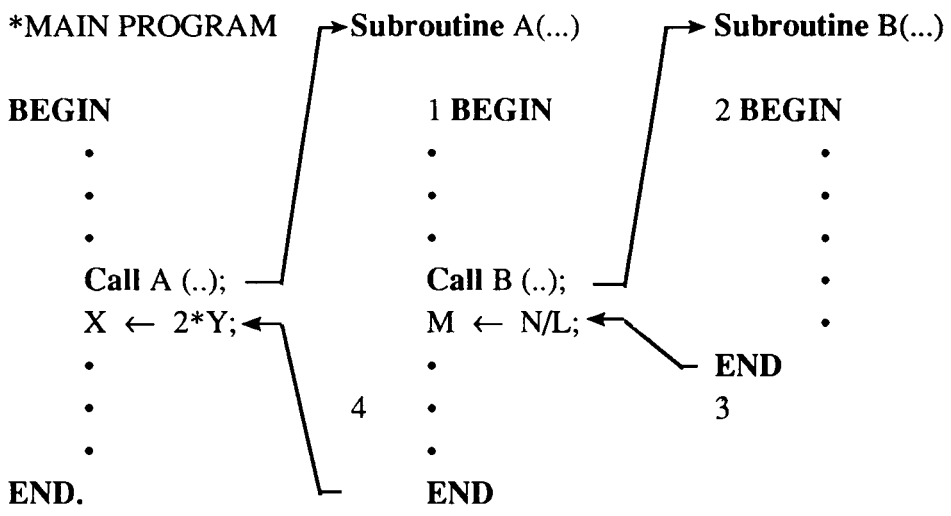
Perlu dibuat sebuah desain Program untuk mengurutkan 3 integer, dari besar ke kecil.

PEMECAHAN:

Kode Semu PROGRAM UNTUK MENGURUTKAN TIGA BILANGAN

```
BEGIN
    Integer INT1,INT2,INT3;
    Input INT1,INT2,INT3;
    If INT1 < INT2 then call SWAPP (INT1,INT2)
    else;
        if INT1 ( INT3 then call
            SWAP < INT2, INT3)
    else;
        output INT1, INT2, INT3
END
*ROUTINE SWAP MENUKAR NILAI DUA INTEGER
Subroutine SWAP (L,M)
BEGIN
    Integer L,M,TEMP,
    TEMP ← L;
    L ← M;
    M ← TEMP
END.
```

Sebuah subprogram dapat "meminta bantuan" dari subprogram lainnya (subrutin atau fungsi lainnya). Hasil suatu rantai(chain) call terdapat di dalam gambar 4-1. Setiap call diselesaikan dengan Statemen untuk segera "return" kepada subrutin yang memenggilnya.



Gambar 4.1. Sebuah "chain call" (panggilan berantai). Panah menunjukkan arah pengendalian Program dalam pelaksanaan; angka-angka menunjukkan rangkaian kegiatannya.

Contoh 4.4

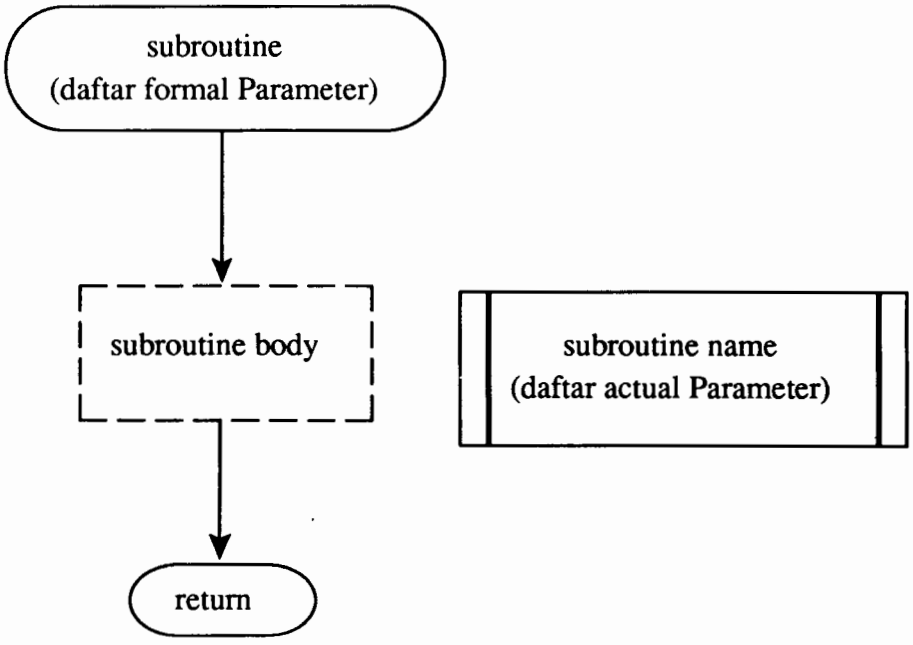
Jika Program kita untuk mengurutkan 3 integer tersebut di atas ingin dipakai secara lebih luas lagi, misalnya untuk menghitung pangkat 3 dari bilangan positif, maka Program tersebut dapat dijadikan subrutin berikut :

```
* URUTAN TRIPEL INTEGER POSITIF
BEGIN
      Integer I,J,K;
      Input I,J,K;
*DATA FLAG (PANJI) ADALAH -1
            While I <> -1 do
            BEGIN
                  Call ORDER(I,J,K);
                  Input I,J,K
            END
END.
```

```
*ROUTINE MENGURUTKAN TIGA INTEGER
Subroutine ORDER(INT1,INT2,INT3)
BEGIN
*SEMUA STATEMENT PROGRAM PENGURUTAN
*CONTOH 4-3 KECUALI STATEMEN INPUT
      •
      •
      •
END

*ROUTINE SWAP MENUKAR DUA NILAI
Subroutine SWAP(L,M)
BEGIN
*SEPERTI CONTOH 4-3
      •
      •
      •
END
```

Usahakan supaya rutin tersebut mengadakan interaksi secara tepat. Sebuah subrutin merupakan Unit Program yang dapat dipakai untuk Program manapun juga, dan memiliki Diagram Alur sendiri (lihat gambar 4-2 (a)). Skema standar yang dipakai untuk memanggil sebuah subrutin di dalam Diagram Alur dari Statemen di dalam subrutin yang menjadi awal pelaksanaan subprogram disebut **entry-point;** Statemen yang mengembalikan pengendalian Program pemanggil disebut **return-point** dari subroutin. Kedua Statemen tersebut merupakan yang pertama dan yang terakhir di dalam subrutin. Beberapa bahasa Pemrograman (seperti FORTRAN 77, COBOL atau PL/I) memperkenankan Pemrogram untuk menetapkan beberapa entry dan return point di dalam sebuah subrutin. Beberapa entry point mempunyai nama dan daftar Parameter sendiri.



(a) Diagram Alur dari sebuah subrutin

(b) Skema untuk memanggil subrutin

Gambar 4.2 Diagram Alur subrutin

4.4 PELAKSANAAN SUBRUTIN DALAM BAHASA PASCAL

Dalam bahasa Pascal, Subrutin dikenal sebagai Prosedur. Prosedur dipanggil dan digunakan di dalam blok Program, dengan menyebutkan judul Prosedur tersebut. Secara umum bentuknya adalah sebagai berikut:

```
PROGRAM Judul-Program;
    PROCEDURE Judul-Prosedur;
    BEGIN
        •
        •
        •
    END;

BEGIN
    •
    •
    •
END.
```

Karena dalam Turbo Pascal, Judul-Program bersifat optional maka bentuk umum Program yang mengandung Prosedur adalah sebagai berikut :

```
PROCEDURE Judul-Program;
BEGIN
    •
    •
    •
END;

BEGIN
    •
    •
    •
END.
```

Sebagai Contoh perhatikan Program yang mengandung Prosedur berikut ini:

```
Program Contoh-Prosedur;
    Procedure Garis;
    Begin
        Writeln ('———— ') ;
    End;

    Begin
        Garis;
        WriteLn ('CONTOH');
        Garis;
    End.
```

Bila Program dijalankan, hasilnya adalah :

CONTOH

Prosedur di atas digunakan untuk membuat garis-garis. Pada Program utama bila dibutuhkan mencetak garis tersebut, kita tinggal menuliskan judul prosedurnya saja, yaitu Garis.

4.4.1 PARAMETER DALAM PROSEDUR PASCAL

Nilai di dalam Modul Program Pascal sifatnya adalah **lokal**, artinya hanya dapat digunakan pada Modul atau unit Program yang bersangkutan saja, tidak dapat digunakan pada Modul atau unit Program yang lainnya.

Contoh:

```
Procedure Kuadrat;
Var
        X,Y : Integer;
Begin
        Write('Nilai X ? ');
        ReadLn(X);
        Y := X * X;
        WriteLn('Nilai Y = ',Y);
End; (* Procedure Kuadrat *)
Begin
    Kuadrat;
End.
```

Bila Program dijalankan :

```
Nilai X ? 5
Nilai Y = 25
```

Pada contoh, variabel X dan Y sifatnya adalah lokal untuk Prosedur Kuadrat, yang artinya hanya dapat digunakan pada Modul tersebut saja. Pada Modul yang lain, variabel tersebut tidak dapat digunakan. Bila digunakan, maka akan salah.

Contoh:

```
Procedure Kuadrat;
Var
    X,Y : Integer;
Begin
    Write ('Nilai X ? ');
    Readln (X);
    Y := X * X;
End; (* Procedure Kuadrat*)
Begin
    Kuadrat;
    WriteLn ('Nilai Y = ',Y);
End.
```

Bila Program akan dikompilasi, akan dideteksi kesalahan oleh kompilator, karena variabel Y yang sifatnya lokal di Prosedur Kuadrat digunakan pada Program

utama. Kesalahan yang ditunjukkan oleh kompilator akan berbentuk pemberitahuan:

Error 31 : unknown identifier

Contoh :

```
Procedure CetakNama;
Begin
    WriteLn (NamaKu);
End;

(— Modul utama ————————)
Var
    NamaKu : String [20];
Begin
    NamaKu := 'Pascal';
    CetakNama;
End.
```

Contoh di atas merupakan contoh Program yang salah karena Modul Prosedur menggunakan variabel lokal untuk Modul utama saja. Pada contoh ini variabel NamaKu juga bersifat lokal untuk Modul utama saja. Pada Modul Prosedur CetakNama, variabel ini digunakan, sehingga akan menghasilkan kesalahan proses. Supaya nilai-nilai variabel dapat digunakan di Modul lainnya yang membutuhkannya. Untuk itu dapat dilakukan dengan beberapa cara, yaitu :

1. Dibuat bersifat global

   Supaya suatu nilai dapat bersifat global, maka harus dideklarasikan di atas Modul yang akan menggunakannya.

   Contoh :

```
Procedure Kesatu;
Begin
    •
    •
End; {— akhir dari procedure Kesatu —}
Var
    A, B : Word

Procedure Kedua;
```

```
Begin
    •
    •
End; {— akhir dari procedure Kedua —}

Procedure Ketiga;
Begin
    •
    •
End;{— akhir dari procedure Ketiga —}

Begin
    •
    •
End.{— akhir dari Modul utama —}
```

Pada contoh di atas, variabel A dan B bersifat global untuk Prosedur Kedua, untuk prosedur Ketiga dan untuk Modul utama, tetapi tidak bersifat global untuk Prosedur Kesatu, sehingga Prosedur Kesatu ini tidak dapat menggunakan variabel-variabel tersebut.

Contoh:

```
Var
    A,B : Word;

Procedure Kesatu;
Begin
    •
    •
End;{— akhir dari procedure Kesatu —}

Procedure Kedua;
Begin
    •
    •
End;{— akhir dari procedure Kedua —}
```

```
Procedure Ketiga;
Begin
    •
    •
End;{— akhir dari procedure Ketiga —}

Begin
    •
    •
End.{— akhir dari Modul utamâ —}
```

Pada contoh di atas , variabel A dan B bersifat global untuk semua Modul, yaitu untuk Prosedur Kesatu, Prosedur Kedua, untuk Prosedur Ketiga dan untuk Modul utama, sehingga dapat digunakan untuk semua Modul ini.

Contoh :

```
Procedure Kesatu;
Var
    C,D : Real;
Begin
    •
    •
End;{— akhir dari procedure Kesatu —}

Var
    A,B : Word;

Procedure Kedua;
Begin
    •
    •
End;{— akhir dari procedure Kedua —}

Procedure Ketiga;
Begin
    •
    •
End;{— akhir dari procedure Ketiga —}
```

```
Begin
    •
    •
End.{akhir — dari Modul utama —}
```

Pada contoh di atas, variabel A dan B bersifat global untuk Prosedur Kedua, untuk Prosedur Ketiga dan untuk Modul utama, tetapi tidak bersifat global untuk Prosedur Kesatu, sehingga Prosedur Kesatu ini tidak dapat menggunakan variabel-variabel tersebut. Sebaliknya variabel C dan D adalah hanya bersifat lokal untuk Prosedur Kesatu saja dan tidak dapat digunakan untuk Modul yang lainnya.

2. Dikirimkan sebagai parameter ke Modul yang membutuhkannya. Parameter yang dikirimkan dari Modul utama ke Modul Prosedur disebut dengan **parameter aktual** (actual parameter) dan parameter yang ada dan dituliskan pada judul Prosedur disebut dengan **parameter formal** (formal parameter). Proses pengiriman data lewat parameter aktual ke parameter formal disebut dengan **parameter passing.** Parameter aktual dan parameter formal harus dengan tipe yang sama. Di dalam Pascal, parameter dapat dikirmkan **secara nilai** (by value) atau **secara acuan** (by reference).

4.4.2 PENGIRIMAN PARAMETER SECARA NILAI DALAM PASCAL

Bila parameter dikirim secara nilai (by value), parameter formal di Prosedur akan berisi nilai yang dikirimkan yang kemudian bersifat lokal di Prosedur. Bila nilai parameter formal di Prosedur tersebut berubah, tidak akan mempengaruhi nilai parameter aktual (nilai parameter aktual tetap, tidak berubah). Pengiriman secara nilai ini merupakan pengiriman searah, yaitu dari parameter aktual ke parameter formal, yang tidak dikirimkan balik dari parameter formal ke parameter aktual. Parameter yang digunakan dengan pengiriman secara nilai ini disebut dengan **parameter nilai** (value parameter).

Contoh :

```
Procedure Hitung(A,B : Integer);
Var
    C :Integer;
Begin
    C:=A + B;
    WriteLn(#10,'Nilai C = ',C);
End;(* Procedure Hitung *)
```

```
Var
    X,Y : Integer;
Begin
    Write('Nilai X ?');
    ReadLn(X);
    Write('Nilai Y ?');
    ReadLn(Y);
    Hitung(X,Y);
End.
```

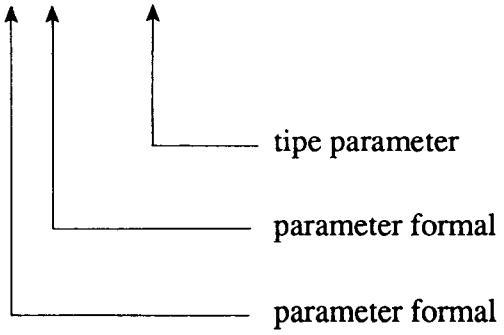
Program di atas merupakan Contoh Pengiriman parameter secara nilai. Bila Program dijalankan :

Nilai X ? 2
Nilai Y ? 3
Nilai C = 5

Penjelasan :

1. Prosedur dimulai dengan deklarasi Prosedur dengan judul Prosedur:

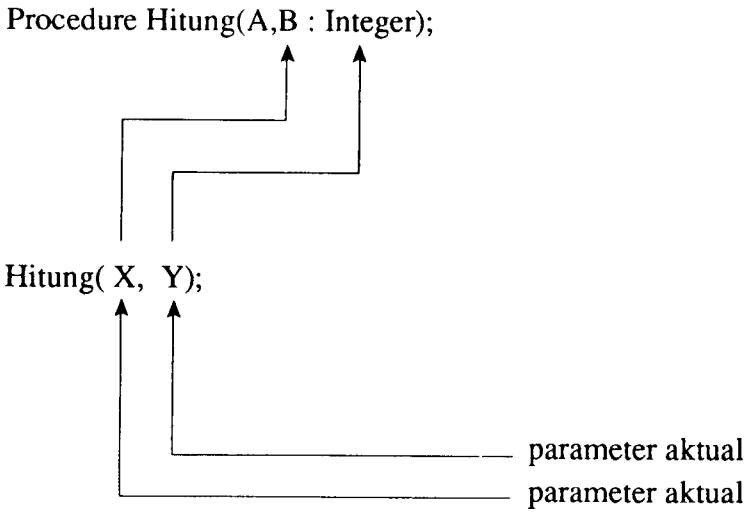
```
Procedure Hitung( A, B : Integer );
```



2. Variabel lokal yang hanya dipergunakan di Prosedur dan tidak termasuk sebagai parameter formal (parameter nilai), harus didefinisikan sendiri, sebagai berikut:

```
Var
    C : Integer;
```

3. Hubungan antara parameter formal di Prosedur dengan parameter aktual di Modul utama adalah seperti berikut :



Terlihat bahwa nilai parameter aktual X dan Y di Modul utama dikirimkan ke parameter formal A dan B di Prosedur. Dengan demikian, nilai parameter A dan B di Prosedur akan berisi nilai yang sama dengan parameter X dan Y di Modul utama.

Contoh :

```
Procedure Hitung(A,B,C : Integer);
Begin
    C:=A + B;
    WriteLn ('A = ',A,'B = ',B,'C = ', C);
End. (* Procedure Hitung*)

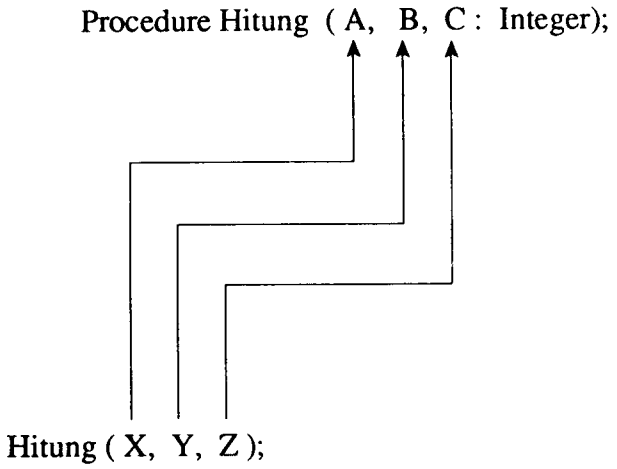
Var
    X,Y,Z :Integer;
Begin
    X:=2;    Y:=3;   Z:=0;
    Hitung(X,Y,Z);
    WriteLn('X =',X,'Y =',Y,'Z = ',Z);
End.
```

Pada Program di atas, Perubahan parameter formal tidak mengubah nilai parameter aktual

Bila Program dijalankan :

A = 2 B = 3 C = 5
x = 2 Y = 3 Z = 0

Hubungan dari pengiriman parameter secara nilai ini adalah sebagai berikut :



Parameter formal A akan terisi nilai yang sama dengan parameter aktual X yaitu benilai 2. Parameter formal B akan terisi nilai yang sama dengan parameter aktual Y, yaitu 3. Parameter formal akan berisi nilai yang sama dengan parameter aktual Z, yaitu 0. Pada modul prosedur, parameter formal C kemudian berganti nilai sebesar nilai A ditambah dengan nilai B, sehingga nilai parameter formal C menjadi sebesar 2 + 3 = 5. Nilai akhir dari parameter formal A, B dan C adalah :

A = 2; B = 3; C = 5

Karena parameter formal dengan pengiriman parameter secara nilai sifatnya lokal, maka perubahan nilai parameter di prosedur tidak akan mempengaruhi nilai parameter aktual di modul lain, sehingga dengan demikian nilai parameter Z tetap tidak terpengaruh oleh perubahan nilai parameter C, jadi masih bernilai 0.

4.4.3 PENGIRIMAN PARAMETER SECARAACUAN/REFERENSI DALAM PASCAL

Bila pengiriman parameter secara acuan (by reference), maka perubahan-perubahan yang terjadi pada nilai parameter formal di prosedur akan mempengaruhi nilai parameter aktual. Parameter ini disebut dengan *variabel parameter* serta dideklarasikan di deklarasi prosedur dengan menggunakan kata cadangan *Var*, sebagai berikut :

PROCEDURE Hitung (VAR A, B, C : Integer);



menunjukkan pengiriman parameter secara acuan

Contoh Pengiriman parameter secara Acuan :

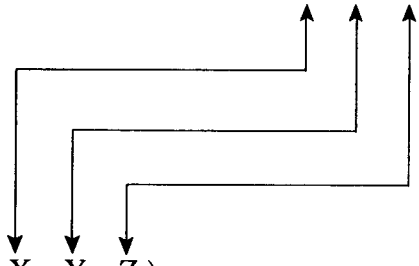
```
Procedure Hitung(Var A,B,C : Integer);
    Begin
        C := A + B;
    End; (*Procedure Hitung*)
    Var
        X,Y,Z:Integer;
    Begin
        X := 2; Y:= 3;
        Hitung (X, Y, Z);
        Writeln ('X = ', X,' Y = ', Y,' Z = ' ,Z );
    End.
```

Bila Program ini dijalankan diperoleh :

x = 2 Y = 3 Z = 5

Hubungan antara parameter formal dengan parameter aktual dengan pengiriman parameter secara acuan ini adalah sebagai berikut :

Procedure Hitung (Var A, B, C : Integer);



Hitung (X, Y, Z);

Terlihat bahwa pengiriman parameter secara acuan merupakan pengiriman dua arah, bolak-balik, sehingga perubahan nilai di parameter formal akan mempengaruhi nilai parameter aktual juga. Pada contoh Program, nilai parameter aktual Z akan mengikuti perubahan nilai dari parameter formal C.

Contoh :

Program berikut digunakan untuk menghitung faktorial, dengan pengiriman parameter secara Acuan.

Kita ingat bahwa :

N! = 1 * 2 * 3 * ...* N

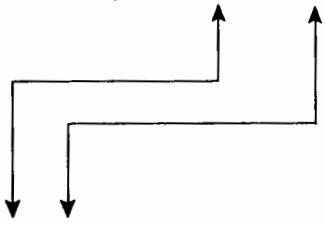
```
Procedure Faktorial(Var Fak,Hasil : Integer);
Var
    I : Integer;
Begin
    Hasil := 1 ;
    For I := 2 To Fak Do
        Hasil : = Hasil * I;
End; (* Procedure Faktorial *)
Var
        N,F : Integer;
Begin
    Write ('Berapa faktorial ? ');
    ReadLn (N);
    Faktorial (N,F);
    WriteLn; WriteLn (N,' Faktorial = ',F);
End.
```

Bila Program ini dijalankan:

```
Berapa faktorial ? 5
5 Faktorial = 120
```

Hubungan antara parameter formal dengan parameter aktual adalah sebagai berikut :

```
Procedur Faktorial (Var  Fak,  Hasil : Integer );
```



```
Faktorial ( N,  F );
```

Contoh:

Banyaknya kombinasi dapat dihitung dengan rumus:

$$C \overset{R}{\underset{N}{\equiv}} \frac{N!}{(N-R)!.R!}$$

N = banyaknya obyek / data yang akan dikombinasikan
R = banyaknya ambilan kombinasi
C = jumlah kombinasi yang terjadi

Misalnya banyak obyek yang akan dikombinasikan adalah sejumlah 3 buah, yakni A, B dan C. Akan dikombinasikan dua-dua (R = 2), maka kombinasi yang akan terjadi adalah :

AB, AC dan BC

berarti jumlah kombinasi yang akan terjadi (C) adalah sebanyak 3 macam. Kalau diperhatikan rumus dari menghitung kombinasi ini adalah menggunakan perhitungan faktorial. Sedang proses menghitung besarnya suatu faktorial telah diberikan pada Program sebelummnya dalam bentuk prosedur. Prosedur menghitung besarnya nilai suatu faktorial tersebut dapat dimanfaatkan di dalam menghitung jumlah kombinasi ini.

Berikut ini Contoh Prosedur Faktorial untuk memghitung kombinasi dengan pengiriman parameter secara acuan

```
Procedure Faktorial ( Var Fak, Hasil : Integer );
Var
    I : Integer;
Begin
    Hasil := l;
    For I :=2 To Fak Do
    Hasi l := Hasil * I ;
End; (" Procedure Faktorial *)
Var
    R,N,NminusR,F1,F2,F3    : Integer;
    C                       : real;
Begin
    Write('Banyaknya data   ?');
    ReadLn(N);
    Write('Banyaknya kombinasi ? ');
    ReadLn(R);
```

```
        N minus R := N - R;
        Faktor(n,F1);
        Faktorial ( N minus R, F2 );
        C := F1 / (F2 * F3);
        WriteLn; WriteLn ('jumlah kombinasi yang terjadi =',C:9:0);
    End.
```

Bila Program di atas dijalankan:

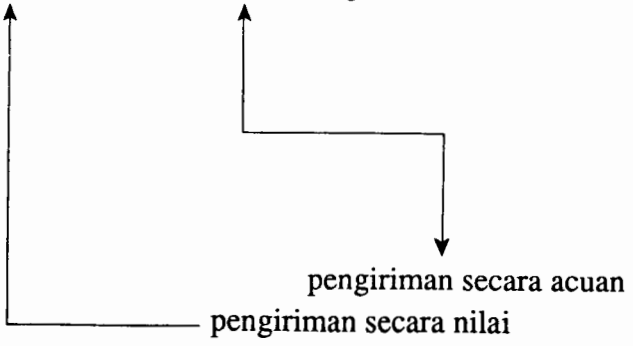
```
Banyaknya data     ? 5
Banyaknya kombinasi ? 3

Jumlah kombinasi yang terjadi = 10
```

CATATAN :

Pengiriman parameter dapat dicampur sebagian secara nilai dan sebagian secara acuan dalam suatu prosedur. Parameter yang hanya dibutuhkan pada prosedur saja dapat dikirimkan secara nilai dan yang ingin dikirimkan balik dapat dilakukan secara acuan, sebagai berikut:

```
Procedure Hitung ( A, B : Integer;  Var C  :  Integer );
```



Contoh: Pengiriman parameter sebagian secara nilai dan sebagian secara acuan

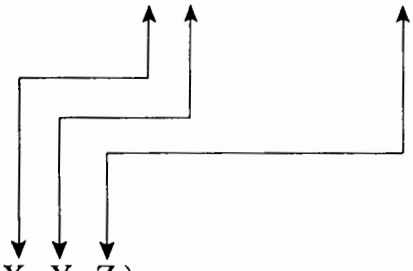
```
Procedure Hitung (A, B : Integer;  Var C : Integer);
    Begin
        C := A + B;
    End; (* Procedure Hitung *)
    Var
        X,Y,Z Integer;
    Begin
        x := 2; Y:= 3;
        Hitung (X,Y,Z);
        WriteLn('X = ',X,' Y = ',Y,' Z = ',Z);
    End.
```

Bila Program ini dijalankan :

x = 2 Y = 3 Z =5

Hubungan antara pengiriman parameter tersebut adalah sebagai berikut :

Procedure Hitung(A, B : Integer; Var C : Integer);



Hitung (X, Y, Z);

Keterangan:

Bila data dengan volume yang besar akan dikirimkan sebagai parameter, maka penggunaan parameter variabel (dikirimkan secara acuan) akan menghemat lokasi memori yang sama (dengan alamat yang sama). Kalau pengiriman parameter secara nilai, yang dikirimkan adalah datanya, sehingga dibutuhkan lokasi memori tersendiri untuk menyimpannya dan dibutuhkan waktu pemrosesan yang lebih lama untuk meletakkan data tersebut di alamat memori lainnya yang terpisah. Akibatnya lebih banyak memakai lokasi memori dan lebih lambat. Karena pengiriman secara nilai ini menggunakan alamat yang tersendiri, maka perubahan nilai pada parameter formal tidak akan mempengaruhi nilai parameter aktual, karena lokasinya berbeda.

4.4.4 TIPE PARAMETER FORMAL DALAM PASCAL

Setelah dibahas bahwa tipe dari parameter formal harus dideklarasikan. Parameter yang diizinkan adalah berupa variabel atau konstanta, sedang pada turbo pascal prosedur dan fungsi tidak diizinkan sebagai parameter formal atau standard pascal mengizinkan. Tipe sederhana, seperti Integer, Byte, Char, Real dan Boolean dapat langsung dipergunakan sebagai tipe dari parameter formal. Sedang tipe yang kompleks seperti misalnya string, array, dan record tidak dapat digunakan langsung sebagai tipe parameter formal, tetapi harus dideklarasikan terlebih dahulu di bagian deklarasi *Type*.

Misalnya penulisan parameter formal tipe larik (array) sebagai berikut :

Procedure Urutkan(Nilai : Array [1..1000] of Integer);

Merupakan penulisan yang salah. Penulisan yang seharusnya adalah dengan mendeklarasikan di bagian Type terlebih dahulu, sebagai berikut :

```
Type
    Elemen = Array [1..1000] of Integer;
Procedure Urutkan (Nilai : Elemen);
```

Demikian juga penulisan berikut ini adalah salah:

```
Procedure HitungPanjang (Nama : string [30] );
```

Penulisan yang benar adalah:

```
Type
    Huruf = String [30];
Procedure HitungPanjang (Nama : Huruf);
```

Untuk pengiriman parameter string secara nilai *(by value)*, panjang parameter formal dan parameter aktual dapat tidak sama. Akan tetapi untuk pengiriman parameter *string* secara acuan *(by reference)* panjangnya harus sama.

Contoh:

Pengiriman parameter string secara nilai, panjangnya dapat tidak sama

```
Type
    Huruf = String[100];
Procedure Hitungpanjang(Nama: Huruf; Var panjang : Integer);
Begin
    Panjang := Length(Nama);
End;
Var
    N : Integer;
    NamaAnda:string[30]
Begin
    Write ('Nama Anda ? ');
    ReadLn (NamaAnda);
    HitungPanjang (NamaAnda,N);
    WriteLn ('Panjang nama Anda adalah ',N, ' karakter' );
End.
```

Bila Program dijalankan

```
Nama Anda ? Endang Sri Dewi
Panjang nama Anda adalah 15 karakter
```

Tampak pada Program bahwa pengiriman parameter *string* secara nilai ini menggunakan panjang yang berbeda, yaitu panjang parameter formal adalah 100 karakter dan panjang karakter aktual adalah 30 karakter dan tidak terjadi kesalahan.

Secara normal, pengiriman parameter *string* secara acuan, panjang parameter formal dan parameter aktual harus sama. Bila panjangnya tidak sama, maka akan terjadi kesalahan. Turbo Pascal menyediakan *compiler directive*, yaitu {$V+} yang mengharuskan panjang parameter formal dan aktual harus sama dalam pengiriman secara acuan dan *compiler directive* {$V-} untuk mengijinkan panjang dari parameter formal dan aktual yang berlainan. Defaultnya adalah {$V+}.

Contoh :

Bila Program sebelumnya untuk menghitung panjang dari suatu nama akan diubah dengan panjang paramater formal dan parameter aktual yang berlainan dalam pengiriman parameter secara acuan, supaya tidak terjadi kesalahan, maka harus ditambahkan *compiler directive* {$V-} sebagai berikut :

```
{$V-}
Type
    Huruf = String[100];
    Procedure Hitung Panjang ( Var Nama : Huruf ; Var Panjang :
    Integer);
Begin
    Panjang:=Length(Nama);
End;
Var
    N : Integer;
    NamaAnda : String[30];
Begin
    Write ('Nama Anda ?');
    ReadLn (NamaAnda);
    HitungPanjang (Nama Anda);
    WriteLn ('Panjang nama Anda adalah ' N, 'karakter' );
End.
```

Bila Program ini dijalankan :

```
Nama Anda ? Endang Sri Dewi
Panjang nama Anda adalah 15 karakter
```

4.4.5 PROSEDUR MEMANGGIL PROSEDUR YANG LAIN DALAM PASCAL

Di dalam suatu prosedur dapat memanggil prosedur yang lain.

Contoh :

Prosedur memanggil prosedur yang lainnya.

```
Procedure Pro1(x1: Integer);
    Begin
        WriteLn ('Nilai X = ' Xl,' ada di prosedur PRO1');
    End;

Procedure Pro2(X2 : Integer);
    Begin
    WriteLn ('Nilai X = ', x2,' ada di prosedur PRO2'); Prol (x2);
End;

Var
    X : integer;
Begin
    X := 5; Pro2(X);
End.
```

Bila Program ini dijalankan, diperoleh :

Nilai X = 5 ada di prosedur PRO2
Nilai X = 5 ada di prosedur PRO1.

Penjelasan :

Pada modul utama, nilai X adalah sebesar 5 dan dikirimkan sebagai parameter ke prosedur Pro2. Pada prosedur Pro2 nilai ini ditampilakn dan pada prosedur ini memanggil prosedur yang lainnya, yaitu prosedur Pro1 juga dengan mengirimkan nilai X2 ke prosedur Pro1. Pada prosedur Pro1 nilai tersebut ditampilkan.

4.4.6 PROSEDUR TERSARANG (NESTED) DALAM PASCAL

Prosedur tersarang (*nested procedure*) adalah prosedur yang berada di dalam prosedur yang lainnya. Bentuk Prosedur tersarang adalah sebagai berikut :

```
PROGRAM

    PROCEDURE

        PROCEDURE

        BEGIN


        END;

    BEGIN


    END;

BEGIN


END.
```

Contoh Program prosedur tersarang:

```
Procedure Kesatu;
    Procedure Kedua;
    Begin
        WriteLn ('Prosedur KEDUA ini ada di dalam prosedur KESATU)
    End;
    Procedure Ketiga;
    Begin
        WriteLn (' Prosedur KETIGA ini juga ada di dalam prosedur
        KESATU');
    End;
Begin
    Writeln ('Ini Prosedur KESATU');
    WriteLn;
```

```
        Kedua;
        Ketiga
End;

Begin
        WriteLn ('Ini modul utama');
        WriteLn;
        Kesatu;
End.
```

Bila Program ini dijalankan, diperoleh :

```
Ini modul utama

Ini Prosedur KESATU
Prosedur KEDUA ini ada di dalam prosedur KESATU
Prosedur KETIGA ini juga ada di dalam prosedur KESATU
```

4.4.7 PROSEDUR MEMANGGIL DIRINYA SENDIRI DALAM PASCAL

Prosedur memanggil dirinya sendiri merupakan suatu prosedur yang memanggil atau menggunakan prosedur juga. Proses dari suatu Program bagian yang memanggil dirinya sendiri dikenal dengan istilah *recursion*. Tidak semua bahasa menyediakan kemampuan untuk melakukan proses *recursion*, tetapi Pascal dapat. Walaupun proses ini merupakan *algorithm* yang baik, tetapi membutuhkan banyak memori, karena setiap kali Program bagian dipanggil oleh dirinya sendiri, sejumlah ruang memori tambahan dibutuhkannya.

Contoh :

Program Prosedur memanggil dirinya sendiri (rekursi):

```
Var
        I : integer;
Procedrue Rekursi;
Begin
        WriteLn ('Pondokcina Depok');
        I : = I + 1;
        If I < 10 Then
                Rekursi;
End;
```

```
Begin
    I : = 1;
    Rekursi;
End.
```

Bila Program ini dijalankan, maka prosedur Rekursi akan diproses sebanyak 9 kali, yaitu dengan cara memanggil dirinya sendiri, sebagai berikut :

Pondokcina Depok
Pondokcina Depok
Pondokcina Depok
Pondokcina Depok
Pondokcina Depok
Pondokcina Depok
Pondokcina Depok
Pondokcina Depok
Pondokcina Depok

4.4.8 ACUAN FORWARD DALAM PASCAL

Acuan *forward (forward reference)* digunakan untuk mendeklarasikan di muka judul prosedur terpisah dari bloknya. Judul prosedur yang berisi parameter terpisah ini merupakan judul yang semestinya dan dideklarasikan dengan menambahkan kata cadangan *Forward*. Blok prosedur ini letaknya terpisah dari judulnya, juga diawali dengan judul prosedur, tetapi hanya mencantumkan judul atau nama prosedurnya saja, tanpa mencantumkan parameter formalnya.

Contoh Program dengan Acuan Forward :

```
Procedure Pro1(Var I : integer); Forward;
    Procedure Pro2 (Var I : integer);
    Begin
        WriteLn ('Prosedur PRO',I);
    End;
    Procedure Pro1;
    Begin
        WriteLn ('Prosedur PRO',1);
    End;
    Var
        I : integer;
```

```
Begin
    I := 1;
    Pro1 (I);
    I := 2;
    Pro2(I);
End.
```

Bila Program ini dijalankan :

Prosedur PRO1
Prosedur PRO2

Tampak pada Program bahwa prosedur PRO1 menggunakan acuan *forward.*

4.4.9 PROSEDUR STANDAR DALAM PASCAL

Turbo Pascal menyediakan prosedur standar yang telah ada dan tidak perlu dibuat sendiri lagi oleh pemakai (lihat Bab 3.10). Pada bagian ini akan dibahas beberapa prosedur standar yang belum dibahas pada bab lainnya, yaitu *Exit, Halt, Move* dan *FillChar.*

Prosedur standar Exit. Prosedur standar ini digunakan untuk keluar dari suatu blok. Bila diletakkan pada Program bagian, prosedur standar *Exit* akan menyebabkan proses di Program bagian berhenti dan proses kembali ke blok Program yang lainnya. Bila berada di Program utama menyebabkan proses Program berhenti. Prosedur standar *Exit* hanya dapat digunakan mulai Turbo Pascal versi 3.0.

Contoh :

```
(* Prosedur standar ini hanya berlaku untuk TURBO Pascal mulai versi 3.0 ke atas*)
Procedure Faktorial(Fak: Integer; Var Hasil : Integer);
    Var
        I : integer;
    Begin
        Hasil := l;
        If Fak = 0 Then
            Exit;
        For I := 2 to Fak DO
            Hasil := Hasil * i;
    End;
    Var
        N, F : integer;
```

```
Begin
    Write('Berapa Faktorial ? ');
    ReadLn (N);
    F := 1;
    Faktorial (N,F);
    WriteLn (N, ' faktorial = ' ,F);
End.
```

Bila progam ini dijalankan, diperoleh:

```
Berapa Faktorial ? 5
5 faktorial = 120
```

Prosedur standar *Exit* tampak sebagai berikut :

```
if Fak = 0 Then
Exit;
```

Maksud dari Statement tersebut adalah untuk mengembalikan proses ke Program utama bila nilai Fak telah sama dengan nol.

Prosedur Standar Halt. Prosedur standar ini digunakan untuk menghentikan proses progam baik di bagan maupun di progam utama.

Contoh :

```
Procedure TampilkanTerus;
Var
    Tekan :Char;
Begin
    Tekan := 'T';
    while (Tekan = 'T') Or (tekan='t') Do
    Begin
        WriteLn ('Pondokcina Depok —> ');
        Write('Tekan < T > untuk Terus < B > untuk berhenti → ' );
        Tekan := Read Key;
        If (Tekan = 'B' ) Or (Tekan = ' ') Then
            Halt;
        WritLn;
    End;
    WriteLn ('Salah !!!'); TampilkanTerus;
End;
Begin
    TampilkanTerus;
End.
```

Bila Program ini dijalankan, diperoleh:

Pondokcina Depok → Tekan <T> untuk Terus <B> untuk berhenti → Pondokcina Depok → Tekan <T> untuk Terus <B> untuk berhenti → Salah !!!

Pondokcina Depok → Tekan <T> untuk Terus <B> untuk berhenti →

Bila Anda tekan 'T' atau 't' maka 'Pondokcina Depok' akan ditampilkan lagi. Kalau Anda menekan 'B' atau 'b' proses akan dihentikan.

Prosedur Standar Move. Bentuk umum :

Move (Var source, dest; count : word)

Prosedur standar ini digunakan untuk menyalinkan suatu blok sebanyak count *byte* memori dari blok dimulai *byte* pertama source dan disalinkan ke *byte* pertama *dest*.

Contoh:

```
Var
    Sumber, Tujuan : string[5];
    Sejunlah :word;
Begin
    Sumber := 'ABCDE';
    Tujan := 'FGHIJ';
    Sejumlah := 4;
    Move (Sumber,Tujuan,Sejumlah);
    WriteLn (Sumber:7,Tujuan:7);
End.
```

Bila Program ini dijalankan, diperoleh :

ABCDE ABCIJ

Prosedur standar Fill Char.Prosedur standar ini digunakan untuk mengisi sejumlah *byte* nilai ke dalam suatu variabel, sebagai berikut :

FillChar (x; count:word;ch)

X adalah variabel yang dapat bertipe apapun yang akan diisi dengan nilai tipe ordinal Ch sebanyak count *byte*.

Contoh Program Pengisian sejumlah byte ke dalam suatu variabel tidak dengan menggunakan prosedur standar Fillchar:

```
Var
    Garis : string[80];
Begin
    Garis:='----------------';
    WriteLn (Garis);
    WriteLn ('Selamat Belajar ?');
    WriteLn (Garis);
End.
```

Pada contoh ini tampak bahwa variabel *string* Garis diisi dengan nilai '-' sebanyak 16 buah. Pengisian variabel dengan nilai ini dapat juga dilakukan dengan menggunakan prosedur standar *Fillchar* sebagai berikut :

```
Var
    Garis : String[8Ø];
Begin
    Fillchar (Garis, Size Of (Garis),'-');
    Garis[0] := #16;

    WriteLn (Garis);
    WriteLn ('Selamat Belajar ?');
    WriteLn (Garis);
End.
```

Dengan menggunakan prosedur standar *Fillchar*, variabel *string* Garis pada Program diisi dengan karakter '-' sebanyak panjangnya. Panjang dari suatu variabel dapat diketahui dengan fungsi standar *SizeOf*. Karena yang dibutuhkan hanya sebanyak 16 karakter saja dari isi variabel Garis, maka panjang dari isi variabel ini perlu diatur kembali. Panjagn dari suatu variabel string tersimpan di posisi ke 9 dari variabel tersebut. Variabel *string* dapat dianggap sebagai suatu larik karakter. Dengan demikian untuk mengatur kembali panjang isi variabel *string* garis, maka dapat dilakukan dengan mengisi elemen ke nol dari *string* tersebut, dengan nilai ASCII 16. Bila Program ini dijalankan akan didapatkan hasil :

```
----------------
Selamat Belajar ?
----------------
```

4.5 FUNGSI

Sebuah Fungsi adalah Subprogram yang namanya perlu diberi nilai untuk memudahkan pemanggilan. Bersamaan pula dapat memakai prameter yang dijalankan meliwati dan membagi lokasi memori untuk mendapatkan beberapa nilai tambahan. Walaupun demikian, adalah lebih baik jika kita memakai sebuah Subrutin kalau membutuhkan beberapa nilai di dalam Subprogram. Sebuah Fungsi diminta melalui pemanggilan nama dan dengan daftar Parameter aktual di dalam suatu Statemen berisi konstanta atau variabel yang dapat memberi nilai. Jenis nilai yang dikembalikan oleh Fungsi tersebut harus sesuai dengan kebutuhan. Pemanggilan Fungsi juga disebut *Referensi Fungsi* dan merupakan komponen dari sebuah Statemen dengan bentuk :

... Nama-Fungsi(daftar Parameter aktual) ...

Parameter di dalam tanda kurung tersebut merupakan sisa dari sebuah Statemen yang mempunyai arti tersendiri.

Contoh 4.5

Jika suatu Fungsi untuk menghitung Akar Kuadrat bilangan positif sudah ditetapkan, yakni Fungsi SQRT( ), maka kita dapat memanfaatkannya ,misalnya :

```
X ← SQRT(Y) – 5*Z
```

ataupun

```
if SQRT(X) ⇐ Y then
    Y ← 2*Z
else
    A ← Z-SQRT(X)
```

Berhubung Referensi Fungsi tersebut diganti oleh sebuah nilai, maka nilainya tidak memiliki nama di dalam Program yang meminta bantuannya. Oleh karena itu ada pembatasan tempat bagi Referensi Fungsi di dalam Program.

Sebuah Subrutin "call" merupakan Statemen yang berdiri sendiri, dan dapat ditempatkan di mana saja; sebaliknya sebuah Referensi Fungsi harus ada nilai yang akan digunakan. Dengan kondisi pemanggilan seperti ini, maka seluruh uraian tentang kegiatan subrutin juga berlaku bagi Fungsi.

Referensi Fungsi dilaksanakan sebagai berikut :

1. Parameter dijalankan, jika ada.
2. Fungsi dilaksanakan seolah-olah merupakan sebuah Program yang berdiri sendiri. Sewaktu dilaksanakan, nilai dari jenis Fungsi yang sama dengan nama fungsinya diberikan kepada nama Fungsi tersebut.

3. Fungsinya akan kembali dari nilai-nilai yang didapat digantikan oleh Referensi Fungsi. Nilai lain dapat dikembalikan melalui Parameter yang bersangkutan atau melalui lokasi memori yang ditempati bersama.

Sebuah Fungsi ditetapkan seperti menetapkan sebuah Subrutin. Perbedaannya adalah bahwa jenis fungsi (yakni jenis nilai yang akan digantikan oleh referensi) harus dinyatakan dalam judul Fungsi.

Contoh 4.6

Problema

Tetapkan bilangan terbesar dari tiga integer positif.

Pemecahan dengan Kode Semu

```
*BILANGAN TERBESAR DARI TRIPEL INTEGER POSITIF
BEGIN
        Integer I,J,K;
        Input I,J,K;
*DATA FLAG ADALAH -1
        while I <> – 1 do
            BEGIN
                Output I,J,K,MAX(I,J,K);
                Input I,J,K
            END
END.
*BILANGAN TERBESAR DITEMUKAN
Integer Function MAX(L,M,N)
BEGIN
        Integer L,M,N;
        if L > M then
            MAX ← L
        else
            MAX ← m;
        if K > MAX then
            MAX ← K
        else
END.
```

Perhatikan :

1. Bahwa di dalam Fungsi itu sendiri diberikan sebuah nilai kepada nilai Fungsi tersebut tanpa isi daftar Parameter.

2. Pelajari perbedaan antara jawaban problema di dalam contoh 4.4 dan jawaban di dalam contoh tersebut di atas.

Sebuah Fungsi dapat memberi Referensi kepada sebuah Subrutin; sebuah Subrutin dapat memanggil sebuah Fungsi. Beberapa bahasa Pemrograman sama sekali tidak memiliki sarana bagi Fungsi tersebut (misalnya COBOL).

Di dalam bahasa yang memiliki sarana bagi Fungsi (seperti ALGOL, FORTRAN, dan PL/I) terdapat dua jenis Fungsi. Yang pertama adalah *Fungsi built-in* (juga disebut intrinsic). Fungsi ini dapat dipakai sebagai referensi tanpa perlu penetapan. Fungsi ini ditulis oleh Pemrogram Sistem (System Programmer), dan dimasukkan oleh Kompilator di dalam Program-Utamanya. Fungsi tersebut merupakan Fungsi yang sering dan biasa dipakai, seperti Fungsi Akar, Logaritma, dan sebagainya. Kode tersebut kadang-kadang disalin begitu saja oleh Kompilator sebagai referensi (juga disebut open-routine, sebagai kebalikan dari closed-routine yang memiliki satu copy untuk digunakan). Yang kedua adalah Fungsi yang ditulis Pemrogram, merupakan Fungsi-Fungsi umum tulisan Pemrogram yang ditetapkan untuk dijadikan Referensi.

4.6 BERKOMUNIKASI DENGAN SEBUAH PROGRAM

Data biasanya dimasukkan ke dalam Subprogram yang akan membantu proses; hasil pelaksanaan Subprogram biasanya harus dimasukkan ke dalam Program Unit yang perlu dibantu.

Cara Utama untuk berkomunikasi antara dua Unit Program yang beroperasi di dalam sebuah "call" ialah dengan "menjalankan" Parameteryang bersangkutan. Parameter aktual terdapat dalam daftar Statemen yang membutuhkan bantuan; dan paremeter formal di dalam judul Subprogram.

Cara berkomunikasi yang lain adalah melalui lokasi memori yang dapat dimanfaatkan oleh Program Unit yang meminta dan memberi bantuan. Lokasi tersebut menghimpun data global yang dapat dimanfaatkan oleh seluruh Subprogram yang berkepentingan. Pemakaian data global untuk berkomunikasi antar Unit Program penting sekali artinya bagi Bahasa yang menggunakan "blok structure" (seperti ALGOL atau PL/I). Bahasa Pemrograman tersebut memuat data Modul di dalam Program, berikut dengan modulasi programnya sendiri.

4.7 MENJALANKAN PARAMETER

Melalui mekanisme gerakan Parameter, maka nilai aktual dihubungkan dengan nilai Parameter formal yang sesuai untuk memindahkan data dari Unit Program pemanggil kepada Unit yang dipanggil.

Hubungan tersebut dapat dilakukan dengan tiga cara yang berbeda, dan menghasilkan output yang berbeda pula.

Cara pengalihan Parameter biasanya berada di luar jangkauan Pemrogram, karena ditetapkan oleh implementasi Bahasa. Walaupun demikian, pengertian tentang mekanisme tersebut mempertinggi keahlian Pemrogram terhadap Bahasa yang bersangkutan, dan dapat memperkecil kesalahan.

4.7.1 PEMANGGILAN MELALUI REFERENSI

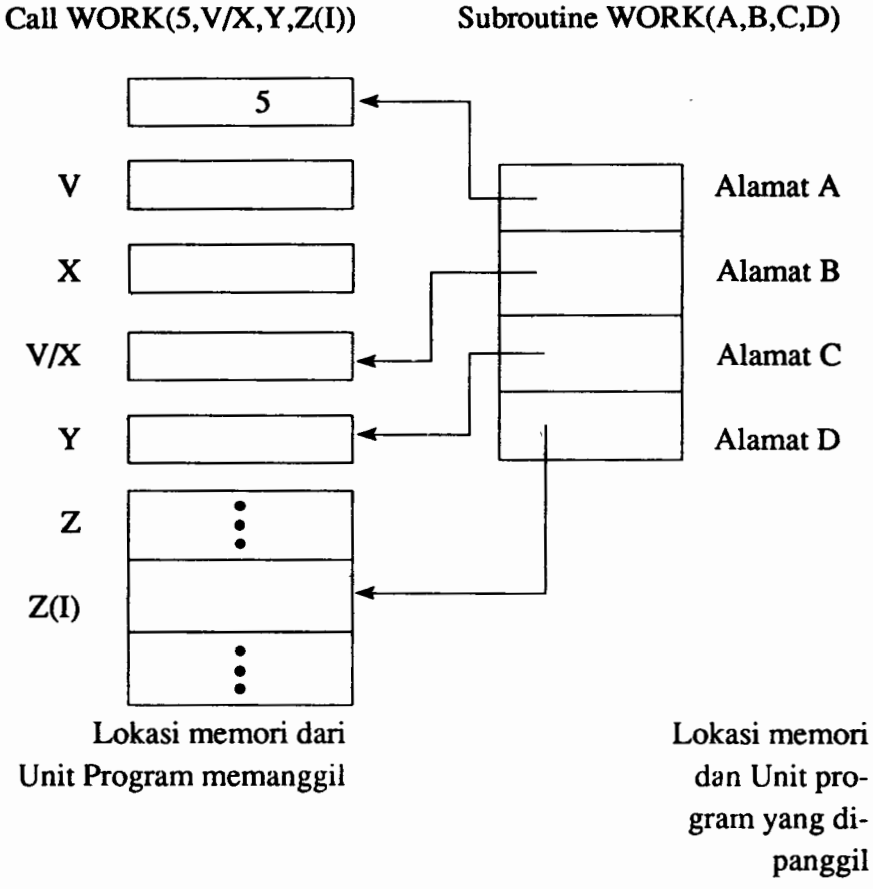
Teknik ini dikenal dengan nama *call by location* atau *by address*, dan paling sering digunakan dalam implementasi pemanggilan Subprogram dalam bahasa COBOL, FORTRAN dan PL/I.

Jika terjadi pemanggilan melalui sebuah referensi, maka alamat dari Parameter aktual dipindahkan ke Subprogram yang dipanggil. Jika parameter aktual tersebut merupakan sebuah ekspressi agar dievaluasi lebih dahulu dan alamat lokasi dari nilai tersebut agar dijalankan, jika Parameter tersebut merupakan sebuah himpunan, maka yang dijalankan adalah elemen pertamanya.

Selanjutnya jika Parameter formal memberi referensi kepada Program yang dipanggil, maka alamat tersebut dipakai untuk membuka lokasi dari Parameter aktual. Dengan demikian maka lokasi-lokasi memori yang mengandung nilai Parameter aktual terbuka bagi Unit Program yang memanggil dan yang dipanggil.

Contoh 4.7

Di dalam gambar 4.3 diperlihatkan lokasi memori yang bertalian dengan Parameter aktual (dari Unit Program yang memanggil) dan Parameter formal (dari Unit Program yang dipanggil). call berikut ini memanggil sebuah subrutin.



GAMBAR 4.3 Pemanggilan dengan referensi. Panah menunjukkan alamat

Perhatikan :

1. Telah dibuat lokasi khusus di dalam memori dari Unit Program pemanggil untuk menampung nilai dari Parameter aktual, yakni V/X.
2. Dua Parameter formal pertama, yakni A dan B jangan dipakai untuk mengembalikan nilai-nilai tersebut kepada Unit Program yang memanggilnya. Jika Program tersebut dilakukan di dalam Unit Program yang dipanggil kepada variabel A dan jika pada waktu kembalinya ke Unit Program pemanggil no 5 dimanfaatkan di sana, maka nilainya bukan 5 lagi.

4.7.2 PEMANGGILAN MELALUI NILAI

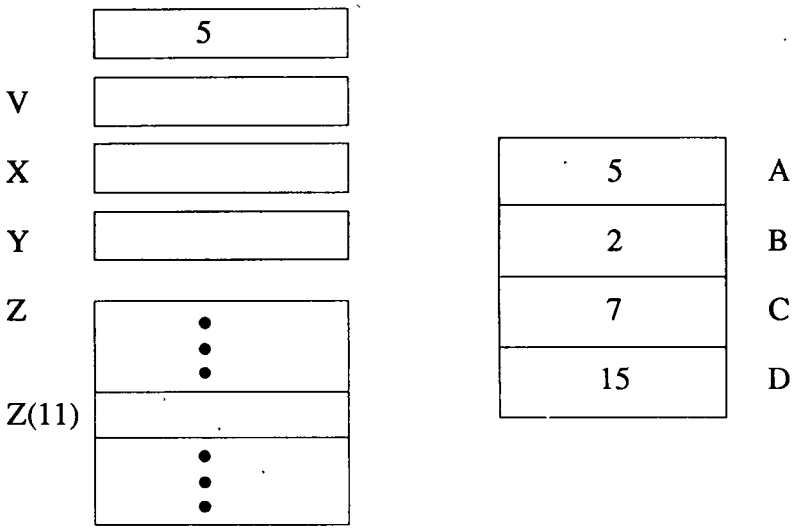
Teknik menjalankan Parameter ini, dapat dibuat oleh seorang Pemrogram ALGOL atau PL/I.

Nilai dari Parameter aktual dijalankan meliwati Unit Program pemanggil. Nilai tersebut ditempatkan di dalam lokasi yang sesuai dengan Parameter formal.

Lokasi memori dari Unit Program pemanggil tidak dapat meliwati Unit Program yang dipanggil. Cara tersebut tidak dipakai, oleh karena itu data harus dikembalikan kepada Unit pemanggil, hanya sesuai untuk menjalankan Parameter input saja.

Contoh 4.8

Program Utama	subroutine :
BEGIN **Integer** I, V, X, Y, Z(100); • • • V ← 4; X ← 2; Y ← 7; I ← 22;Z(I) ← 15; **Call WORK**(5,V/X,Y,Z(I)) • • • **END**	Subroutine WORK(A,B,C,D) **BEGIN** **Integer** A,B,C,D; • • • **END**



GAMBAR 4.4 Pemanggilan melalui nilai

4.7.3 PEMANGGILAN MELALUI NAMA

Cara tersebut dapat dipakai dalam bahasa ALGOL. Prosesnya disebut *copy rule*, yakni sebuah Parameter formal diganti dengan teks oleh Parameter aktual yang sesuai.

Contoh 4.9

Call :

Call WORK(5,V/X,Y,Z(I))

membantu subrutin :

Subroutine WORK(A,B,C,D)
BEGIN

C ← B - A;
D ← C + A;

END

Seluruh variabel adalah dari jenis integer. Sewaktu Subrutin WORK sedang dilaksanakan, maka penugasan berikut ini akan dilaksanakan (bukan WORK di dalam body subrutin) :

Y <— V/X - 5;
Z(I) <— Y + 5;

Pada waktu dilakukan penggantian teks maka rumusan tersebut diberi tanda kurung supaya hasilnya mengandung sebuah arti tertentu. Hal tersebut dapat dilakukan oleh Subprogram khusus yang membutuhkan proses bantuan disebut *thunk*, yang ditetapkan oleh Program penterjemah. Thunk tersebut mengevaluasi Parameter aktual yang nilainya digantikan oleh Parameter formal. Pemanggilan melalui nama jarang dilakukan untuk melaksanakan Statemen dalam bahasa Pemrograman.

4.7.4 MEMBANDINGKAN CARA MENJALANKAN PARAMETER

Pemrogram sering kali tidak dapat mengendalikan "perjalanan" Parameter, tetapi ada beberapa bahasa Pemrograman yang dapat melakukan pengendalian tersebut. Di dalam bahasa ALGOL, misalnya dalam subrutin yang sama, beberapa Parameter dapat dipanggil melalui namanya, sedangkan yang lain dapat dipanggil melalui nilainya.

Pemanggilan melalui nama merupakan cara yang paling banyak digunakan, tetapi pelaksanaannya agak rumit. Pemanggilan melalui nilai tidak begitu umum, karena tidak ada data yang dapat dikembalikan kepada Program pemanggil. Di samping itu juga tidak mungkin memindahkan seluruh himpunan melalui nilainya, sedangkan sebuah alamat sudah cukup bagi pemanggilan melalui referensi. Walaupun demikian cara pemanggilan tersebut membutuhkan modifikasi jika ingin dijalankan meliwati Program yang dipanggil dalam bentuk sebuah ekspressi, bukan nilainya.

Contoh di bawah ini memperlihatkan bahwa cara pemanggilan yang berlainan akan membuahkan hasil yang berbeda pula.

Contoh 4.10

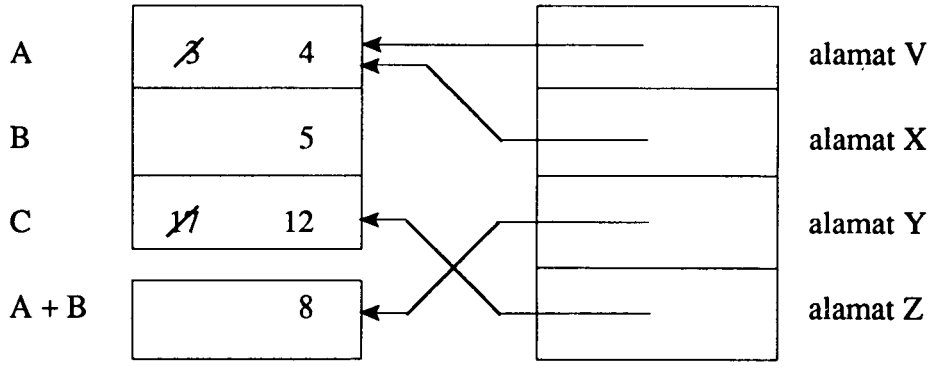
Ada Program sebagai berikut :

```
BEGIN
    Integer A, B, C;
    A ← 3; B ← 5; C ←17;
    Call SUBR (A, A, A + B, C);
    Output C
END
```

```
Subroutine SUBR (V, X, Y, Z)
BEGIN
    Integer V, X, Y, Z;
    V ← V + 1;
    Z ← X + Y
END
```

Nilai yang dapat diperlihatkan sebagai variabel C di dalam Program utamanya tergantung dari mekanisme yang dianut untuk menjalankan Parameter (lihat skema di bawah ini).

1. Pemanggilan melalui referensi, hasilnya diperlihatkan di dalam Gambar 4.5



Gambar 4.5 C = 12

2. Pemanggilan melalui nilai
   a) Panggilan murni melalui nilai C tidak diubah, karena C = 17
   b) panggilan melalui nilai hasil, efeknya diperlihatkan pada Gambar 4.6

3. Pemanggilan melalui nama Sesuai dengan "copy rule"

A ← A + 1	A menjadi 4
C ← A + A + B	C menjadi 4 + 4 + 5
	C = 13

4.8 LOKASI MEMORI YANG DIGUNAKAN BERSAMA

Cara lain untuk berkomunikasi antar Unit Program ialah dengan menggunakan lokasi memori yang memuat data bersama. Lokasi tersebut berisi data global.

Di dalam kelompok tertentu dari bahasa Pemrograman lanjutan yang disebut bahasa 'block structure' (ALGOL dan PL/I) ada sebuah fasilitas untuk modulasi ruang lingkup variabel-variabel. Berarti bahwa rangkaian Statemen yang mengandung suatu variabel dapat dibatasi oleh Pemrogram. Di dalam kelompok bahasa tersebut rangkaian Statemen dapat diterjemahkan tersendiri (ada Subprogram intern dan extern). Bagian-bagian tersebut merupakan bagian integral dari seluruh Program yang memanfaatkan sejumlah variabel dengan bagian yang berbeda.

4.8.1 DATA LOKAL DAN GLOBAL

Data lokal (variabel dan himpunan) merupakan data item yang terdapat di dalam beberapa Unit Program yang menganggap data tersebut sebagai lokal. Lokasi-lokasi memori yang memuat data tersebut digunakan bersama oleh Unit-Unit Program tersebut. Unit-Unit dapat memanfaatkan isi dan mengadakan modifikasi seperlunya.

Lebih baik menggunakan data (daripada Parameter) untuk komunikasi antar Unit Program dilihat dari efisiensi operasional, apabila :

* sejumlah data harus dapat digunakan bersama oleh berbagai Unit Program
* himpunan juga harus dapat digunakan bersama, dan pemanggilan tidak dilaksanakan berdasarkan referensi.

Walaupun demikian perlu ditekankan bahwa Parameter yang dijalankan sering membantu kepercayaan terhadap Program yang bersangkutan.

4.8.2 DATA GLOBAL DI DALAM BAHASA YANG TIDAK MEMILIKI BLOCK STRUCTURE

Bahasa Pemrograman yang tidak memiliki block structure seperti FORTRAN, menggunakan cara yang sederhana dalam mengendalikan ruang lingkup variabel dan himpunan di dalam Program.

Di dalam bahasa tersebut sebuah variabel atau himpunan bersifat lokal terhadap Unit Program yang memuatnya, kecuali dinyatakan menjadi bagian dari Unit yang seharusnya berstatus global. Dengan demikian, di dalam bahasa FORTRAN terdapat suatu Statemen umum di dalam seluruh Unit yang sama-sama menggunakan suatu areal data tertentu.

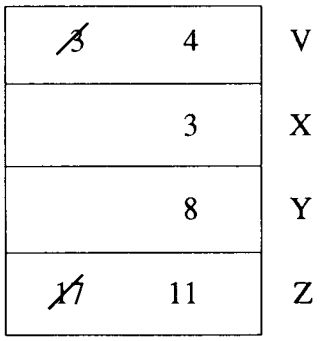
Contoh 4.11

Dalam sebuah Bhasa non-lock structure, terdapat sebuah deklarasi di dalam suatu Program tertentu :

```
Program Utama  :
        Global I,   ARRAY  (100)
Subprogram A   :
        Global K,  SCORES  (100)
Subprogram B   :
        Global M,  VALUES  (100)
```

Lokasi memori yang relevan diperlihatkan di dalam Gambar 4.6.

Lokasi memori dari subrutin



GAMBAR 4.6

Variabel I diketahui sebagai K di dalam Subprogram A dan sebagai M di dalam Subprogram B; himpunan ARRAY sebagai SCORES dan VALUES.

Dengan asumsi bahwa Subprogram X didalam Program tersebut tidak memuat deklarasi Global, maka tidak dapat memasuki lokasi-lokasi tersebut di atas.

4.8.3 BAHASA PEMROGRAMAN BLOCK STRUCTURE

Bahasa Pemrograman seperti itu merupakan bahasa Pemrograman Tingkat Tinggi yang memindahkan pengendalian ruang lingkup dari variabel-variabel dan himpunan; susunannya dapat dimuat dalam bentuk Statemen campuran di dalam Program.

Suatu Statemen campuran dengan bentuk **BEGIN-END** dan diisi dengan deklarasi tertentu disebut **block.**

Sebuah block tersusun dari sejumlah Statemen, dan mungkin juga tersusun dari block-block lain.

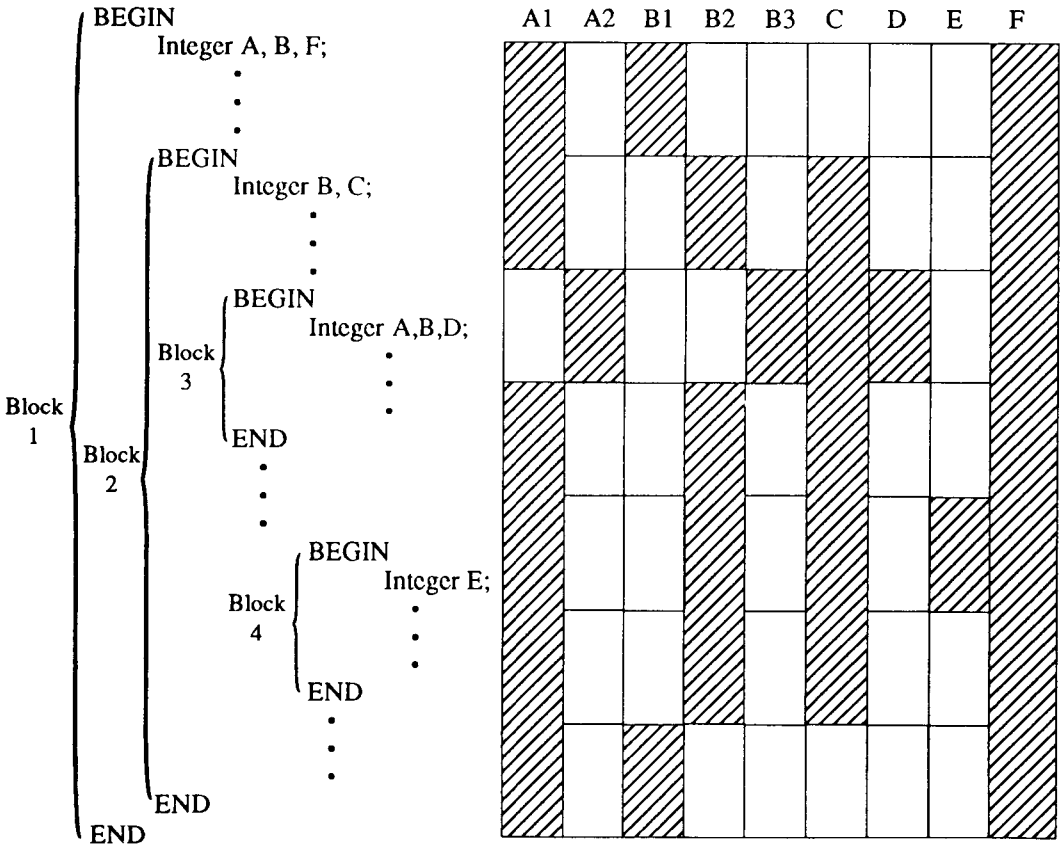
Di dalam Program yang ditulis dalam salah satu block, bahasa misalnya ALGOL atau PL/I, dapat memakai variabel yang berada di dalam ruang lingkup dari deklarasinya.

Ruang lingkup dari variabel dan himpunan ditetapkan oleh Statemen terakhir berikut ini. Ruang lingkup dari variabel mencakup block yang memuat deklarasinya dan seluruh block yang berada di dalamnya, kecuali block-block yang muat deklarasi dengan nama yang sama. Variabel seperti itu sebenarnya tidak sama walaupun kedua variabel tersebut mempunyai nama yang sama, tetapi tidak mempunyai hubungan sama sekali.

Contoh 4.12

Ruang lingkup dari variabel-variabel ini dapat terlihat pada gambar 4.7. Perhatikan bahwa nama B di dalam Program tersebut dipakai untuk 3 buah variabel yang berlainan (ditandai dengan B1, B2, dan B3). Variabel F bersifat global terhadap seluruh Program

Ruang lingkup dari variabel-variabel



GAMBAR 4.7

Bahasa Pemrograman yang block structure dilaksanakan dengan sistem alokasi memori yang dinamis. Memori tersedia bagi variabel, dan himpunan dikala Program sedang dilaksanakan. Jika sebuah variabel sudah ditetapkan maka memori menerima alokasi dari variabel sudah ditetapkan maka memori menerima alokasi dari variabel tersebut; memori baru bebas ketika ruang lingkup dari variabel tersebut sudah habis terpakai karena variabel tersebut akan hilang.

Selanjutnya bagi bahasa Pemrograman yang tidak memiliki susunan Statemen seperti itu (misalnya FORTRAN) maka lokasi-lokasi memori dialokasikan kepada seluruh variabel dan himpunan dikala proses penterjemahan sedang berlangsung, sebelum awal pelaksanaan Program. Sistem tersebut dinamakan **alokasi memori yang statis**. Di dalam bahasa Pemrograman yang block structured, sebuah Subprogram dapat berupa block lain di dalam Program. Subprogram seperti itu juga disebut *internal Subprogram*. Sebuah internal Subprogram menggunakan areal memori bersama dengan block Program yang memuatnya, dan oleh karenanya tidak diterjemahkan secara terpisah. Seluruh variabel dan himpunan yang ruang lingkupnya mencakup Program-Program serupa itu bersifat global terhadapnya.

Subprogram internal hanya terdapat di dalam bahasa ALGOL, sedangkan bahasa PL/I memiliki fasilitas-fasilitas baik untuk internal maupun untuk external Subprogram. Subprogram-Subprogram internal tersebut menggunakan seluruh lokasi data bersama dengan Program utamanya dan tidak dapat dikerjakan dengan Parameter.

Dengan demikian di dalam bahasa yang block structured sebuah Program dapat disesuaikan dengan data masukan. Dalam sebuah block hanya dapat dimuat data global dan lokal saja.

Sebagian besar variabel di dalam sebuah Program hanya digunakan di dalam kelompok Statemen tertentu. Jika kelompok tersebut dijadikan sebuah block, dengan variabel-variabel lokal di dalamnya maka variabel-variabel tersebut :

* tidak dapat dipakai di luar block tersebut sehingga kepercayaan kita terhadap Program yang bersangkutan semakin bertambah.
* dapat diberi nama tersendiri.
* hanya terdapat di dalam block tersebut dan tersedia ruang yang efesien, karena alokasi memori bersifat dinamis.

4.9 REKURSIF DI DALAM PEMROGRAMAN

Solusi terhadap problemaa tertentu sebagian dinyatakan dalam bahasa problemanya. Algoritma solusi seperti itu disebut rekursi, atau memakai definisi rekursif. Sebuah batasan rekursif merupakan sebuah prosedur perhitungan yang terdiri dari :

(1) ketentuan umum untuk mencapai hasil, bagi sebagian besar Parameter; ketentuan tersebut dinyatakan dalam bahasanya sendiri.

(2) Hasil nya.. ..agi beberapa nilai Parameter (tanpa unsur tersebut maka definisi rekursif tersebut akan beredar terus).

Contoh 4.13

Untuk menghitung faktorial N! dari integer positif N kita dapat memakai definisi rekursif berikut ini :

faktor (0) = 1

faktor (N) = N * faktor (N – 1), untuk N > 0

Definisi rekursif tersebut merupakan cara yang biasa dipakai untuk mendapatkan jawaban terhadap problema-problema tertentu. Oleh karena itu sebaiknya juga memakai rekursif di dalam bahasa Pemrograman. Rekursif di dalam Pemrograman menandakan bahwa :

(1) Subprogram dapat memanggil dirinya sendiri (**rekursi langsung**).

(2) pemanggilan Subprogram secara berantai dapat dibenarkan untuk menghasilkan sebuah lingkungan tertutup (misalnya subrutin A memanggil subrutin B yang memanggil lagi subrutin C yang memanggil A), disebut **rekursi tidak langsung.**

Mengapa ada bahasa-bahasa Pemrograman tertentu (seperti FORTRAN dan COBOL) yang melarang rekursi Subprogram?

Ketika sebuah Subprogram dipanggil, lokasi-lokasi tertentu disebut rekaman yang mengaktifkan Subprogram, dipakai untuk memasukkan nilai dari variabel lokal, dan Parameter yang dijalankan (atau alamat dari Parameter) serta mengembalikan Subprogram.

Di dalam bahasa Pemrograman yang memakai alokasi memori statis, unsur penterjemah akan menyediakan lokasi tersebut untuk sebuah Subprogram. Jika sebuah Subprogram maka isi dari lokasi tersebut akan terlampau penuh.

Jika suatu bahasa Pemrograman memperbolehkan rekursi tersebut perlu diadakan alokasi memori dinamis. Bahasa ALGOL dan PL/I memperbolehkannya. Jika suatu Subprogram memanggil dirinya sendiri maka terbentuklah sebuah "rekaman" pengaktif baru. Pengembalian Subprogram harus subrutin A saling memanggil, maka harus memakai "rekaman" pengaktif selama pengembalian sejumlah Subprogram dengan sistem LIFO (yang terakhir dikeluarkan lebih dahulu). "Rekaman" pengaktif ditempatkan di dalam memori dengan sistem LIFO yang membentuk struktur data disebut "stack"(tumpukan).

Jika terjadi pemanggilan, maka sebuah rekaman pengaktif baru ditempatkan di atas "tumpukan" tersebut; jika terjadi pengembalian (rekursi menurun) rekaman pengaktif yang terletak paling atas dipakai sebagai sumber nilai variabel dan alamat pengembalian. Rekaman-rekaman tersebut secara bersama-sama dilakukan dahulu dari tempat pemanggilan terakhir (bandingkan dengan gambar 4-1 dan bayangkan ada sejumlah copy dari gantian dikeluarkan dari tumpukan tersebut). Rekaman yang terakhir akan mengembalikan Subprogram ketempatnya semula.

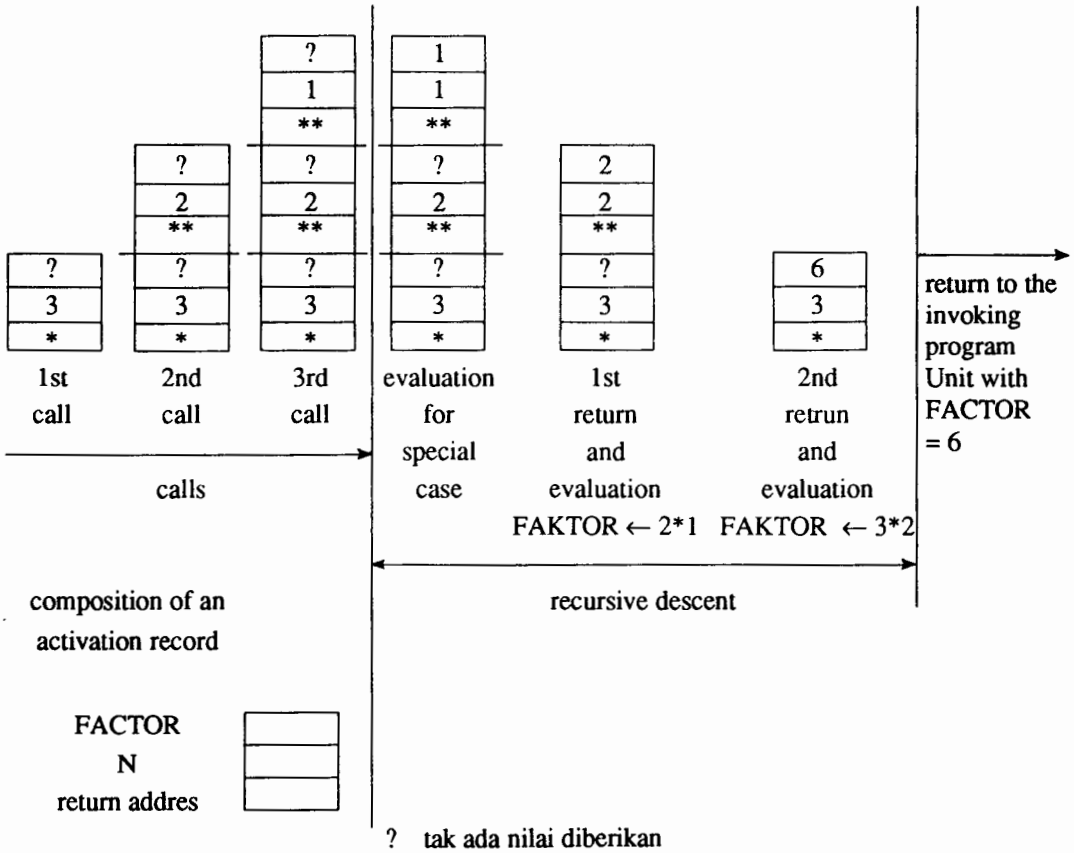
Contoh berikut ini memperlihatkan Fungsi rekursi dari FAKTOR (faktorial N!) dan dinamika dari waktu-susut sebuah "stack"

Contoh 4.14

Kode Semu PERHITUNGAN REKURSIF DARI FAKTOR N > = 0 :

```
Integer Function FACTOR (N)
BEGIN
    Interger N;
    if N = 0 then
        FACTOR <- 1
    else
        FACTOR <- N * FACTOR(N -1)
END •
```

Di dalam Gambar 4.8 diperlihatkan isi sebuah "stock" ketika dilaksanakan Fungsi N = 3. Asumsi pemanggilan melalui nilai.



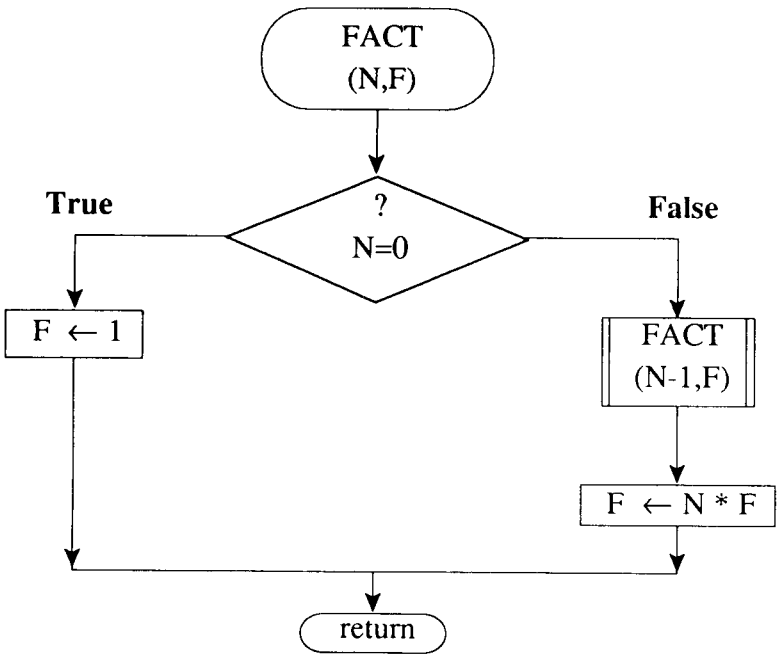
? tak ada nilai diberikan

* alamat Statemen Program pemanggilan kepadanya control k kembali

** alamat Statemen FACTOR ← N * FACTOR(N-1)

Gambar 4. 8

Subprogram tersebut dapat dilaksanakan sebagai subrutin FACT(N,F), seperti diperlihatkan dalam gambar 4.9.



Gambar 4.9

Jawaban rekursif terhadap suatu problema merupakan sebuah alternatif jawaban berulang dari pelaksanaan ulang serangkaian Statemen dengan sebuah loop. Di dalam contoh 4-15 terdapat sebuah Algoritma rekursif sebagai alternatif dari jawaban berulang seperti dibahas di dalam contoh 3-10.

Contoh 4.15

```
Kode Semu BENTUK REKURSIF DARI ALGORITMA ARITMETIKA :
* DIPAKAI UNTUK MENETAPKAN KELIPATAN PERSEKUTUAN
  TERBESAR
* DARI DUA INTEGER
Fungsi Integer KPB(M,N)
BEGIN
        Integer M,N,REM;
*TUKAR BILANGAN KALAU PERLU
        if M < N then
            KPB ← KPB(N,M)   *!*
```

```
*PROSES REKURSIF
        else
            BEGIN
                REM ← M - N * (M ÷ );
            if REM = 0 then
                KPB ← N
            else
                KPB <— KPB(N,REM)
            END
END.
```

Perhatikan :

1. Perhatikan betapa miripnya bentuk ekspresi tersebut dengan uraian tulisan dari Algoritma di dalam contoh 3-10.
2. Perhatikan persamaan antara Parameter aktual dan formal jika terjadi pertukaran :

   N sama dengan M; dan M dengan N

Algoritma rekursif digunakan karena ekspresinya jelas, dan mudah dibaca, walaupun demikian umumnya lebih sulit daripada bentuk-bentuk berulang karena pelaksanaannya lebih cepat dan lebih efisien menggunakan ruang memori karena memakai "stack" untuk rekaman pengaktif sebagai Subprogram. Bahasa-bahasa Pemrograman (seperti FORTRAN) yang tidak memiliki fasilitas untuk rekursif, dapat memekai sistem rekursi dengan memasukkan "stack rutin" ke dalam Program. Ukuran maksimum dari stack tersebut harus diketahui ketika Program ditulis.

4.10 PEMROGRAMAN MODULAR DAN MANFAATNYA

Subprogram yang bersifat external merupakan Modul yang berdiri sendiri, dan dapat membentuk Program besar. Pemrograman Modular merupakan suatu disain Program yang dihasilkan dari proses identifikasi, pembatasan dan pelaksanaan Modul yang akan membentuk Program bersangkutan. Baik sekali dilaksanakan dari atas ke bawah dengan urutan spesifikasi tugas-tugas proyek melalui penyempurnaan Algoritma yang bersangkutan.

Dengan demikian pada langkah-langkah pertama dari proses penyempurnaan algoritma diberi identitas kepada Modul Utama dari Program. Ketika diberi kode-kode tulisan diperlukan dikomposisi Modul dari Program yang bersangkutan. Selanjutnya dapat disimpulkan bahwa sebuah Program yang komplex (sering disebut sistem) memiliki sebuah struktur hirakhis dengan Program Utama sebagai puncaknya dan dengan Modul pembantu untuk melaksanakan tugas-tugas tertentu.

Lebih baik jika setiap Modul dibuatkan uraian tentang Fungsi tunggal yang harus dilaksanakannya. Semakin rendah tingkatan suatu Modul semakin sempit fungsinya. Hal-hal tersebut akan memudahkan pelaksanaan dan pemeliharaan Program (jika Program membutuhkan modifikasi cukup merubah Modul yang melasanakan Fungsi-Fungsi tertentu saja).

Proses desain dari Modul "top-down" diperlihatkan di dalam contoh di bawah ini, yang sekaligus memperlihatkan dua alat penting, yakni *Struktur Arus* dan *Tabel Transaksi*.

Contoh 4.16

Problema:

Kita harus membuat desain dari sebuah Program untuk pemeliharaan periodik sebuah Berkas induk (Master file). Dengan kata lain, membuat up-to-date file yang dihimpun selama waktu tertentu berdasarkan file transaksi yang harus dimasukkan ke dalam file induk tersebut. Dengan demikian, kita perlu menyusun sebuah master file baru.

Penjelasan dan Analisis

Sebuah Master file (misalnya Daftar Gaji sebuah perusahaan) terdiri dari catatan/data Pegawai. Setiap data terdiri sejumlah "field" (misalnya nama pegawai, upah mingguan dan sebagainya) yang masing-masing memberi identitas di dalam file. "Field" tersebut dinamakan KEY (kunci) dari file. Master file disusun berdasarkan key tersebut.

Data dari file transaksi juga diidentifikasi dengan key yang sama. Dengan demikian, sebuah file transaksi untuk memelihara daftar gaji dapat memuat juga data dari pegawai baru. File transaksi tersebut secara umum memuat tiga jenis data untuk dibuat up-to-date :

- modifikasi data dari master file (misal karena gaji pegawai dinaikkan);
- mengeluarkan (misal karena pegawai dipindahkan);
- menyisipkan (misal ada pegawai baru).

Sifat dari pembaruan data tersebut merupakan salah satu "field " yang akan diproses.

Sistem yang akan dirancang sebenarnya sederhana saja, master-file yang kecil dibaca seperlunya dan dimasukkan ke dalam memeori. Berhubung data di dalam file transaksi hanya sedikit, maka isinya tidak tersusun. Dekomposisi Modul dengan sistem "top-down"

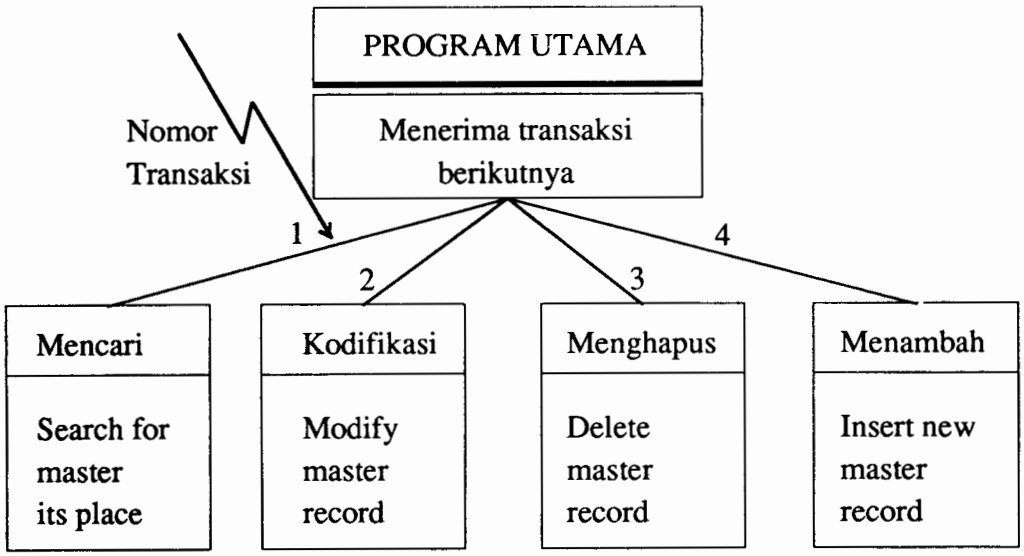
1. Uraian awal dari Algoritma?

```
Kode Semu :
*MEMELIHARA SUSUNAN MASTER FILE
BEGIN
    Input master file;
    Proses seluruh transaksi terhadap master file;
    Output master file baru
END
```

2. Perbaikan:

```
*MEMELIHARA SUSUNAN MASTER FILE
BEGIN
    Input master file;
*PROSES SELURUH TRANSAKSI TERHADAP MASTER FILE
    While ada transaksi do
        BEGIN
            Input  transaksi berikutnya;
            Cari  master record dengan key
            yang disebutkan oleh transaksi;
            if transaksi adalah Modify then
                Modify master record
            else
                if transaksi adalah Delete then
                    Delete master record
                else
                    *TRANSAKSI ADALAH Insert*
                    Insert new master record
        END;
    Output new master file
END.
```

Kini kita mengetahui bahwa kita membutuhkan lima Modul di dalam Program. Modul tersebut diperlihatkan di dalam Diagram Alur gambar 4-10. Sewaktu kita mengadakan perbaikan langkah-langkah instruksi akan terasa adanya kebutuhan untuk membuat Modul pembantu. Perlihatkan bahwa untuk Modul pencarian kita dapat memakai Algoritma binar (contoh 3-17).



Gamber 4.10 Struktur arus

Sewaktu disain mengalami kemajuan agar menetapkan transaksi dari Modul tersebut (yakni pertukaran data Parameter dan global block yang dipakai). Pekerjaan tersebut akan lebih mudah jika memakai tabel berikut ini :

Nomor Transaksi	input Parameter	output Parameter	dipakai global block
Sesuai dengan nomor dalam struktur arus	mengikuti arus ke bawah	mengikuti arus ke atas	sebut nama bloknya

4.11 RANGKUMAN

Pemrograman Modular memberikan manfaat berikut ini (sebagai alternatif terhadap sistem "straight-line code");

1. Melihat Program sebagai sturuktur hirarkhis dari sejumlah Modul yang bekerja sesuai dengan desain tugas, dan lebih mudah dikerjakan.
2. Produk dari desain seperti itu lebih mudah dipahami dan dimodifikasi.
3. Organisasi desain tersebut adalah lebih sederhana, karena dapat dijadikan subbagian.
4. Sebuah Modul dapat dimanafaatkan untuk berbagai tempat di dalam Program yang sama dan beberapa Program lainnya. Oleh karena itu, Modul harus memiliki Fungsi yang terperinci dengan jelas.
5. Modul dapat dikompilasikan dan dicoba secara terpisah. Keadaan tersebut dapat memudahkan pelaksanaan Program.
6. Jika sebuah Program yang terdiri dari sejumlah Modul harus dimodifikasi (sering terjadi di dalam kegiatan Pemrograman), maka kita cukup mengubah Modul tertentu saja.

Pemrograman Modular dapat menghemat waktu, ruang memori dan proses kompilasi. Produknya bisa lebih dipercaya, karena memakai Modul kecil yang lebih mudah dicoba.

Biayanya agak tinggi karena waktu pelaksanaan Program lebih lama sebagai akibat dari pemanggilan Subprogram.