Bab 7
PEMROGRAMAN MODULAR

Dalam suatu pengembangan perangkat lunak, pemrograman adalah salah satu tahap untuk mengimplementasikan penyelesaian masalah tertentu dengan suatu bahasa pemrograman. Penyusunan program yang terstruktur merupakan salah satu syarat program yang baik. Terstruktur berarti memiliki rancangan yang sistematis, mudah dibaca dan , mudah dibetulkan jika ada kesalahan serta mempunyai alur yang jelas (tidak loncat-loncat). Oleh karena itu, penggunaan perintah GOTO harus dihindari.

Salah satu metode penyusunan program terstruktur adalah pemrograman modular. Dengan metode ini, suatu permasalahan yang besar dan kompleks dan dipecah-pecah menjadi beberapa modul sehingga menjadi lebih sederhana. Modul-modul mandiri tersebut biasa dikenal dengan sebutan subroutine. Suatu subroutine pada bahasa BASIC dapat berupa subprogram, procedure maupun function.

Subroutine

Subroutine dibuat dengan beberapa tujuan, yaitu sebagai berikut ini :

1. Proses yang sering terjadi berulang-ulang dikelompokkan sebagai satu subroutine, sehingga bagian tersebut tidak harus ditulis secara berulang-ulang. Dengan demikian pembuatan program akan lebih mudah dan program tidak terlalu panjang.
2. Subroutine perlu dibuat agar program menjadi terstruktur. Salah satu cara agar suatu program menjadi terstruktur adalah dengan cara memecahnya menjadi bagian-bagian yang lebih sederhana dan dapat dijadikan dalam bentuk subroutine.

Untuk membuat program dengan teknik pemrograman modular dalam bahasa BASIC dapat digunakan perintah GOSUB..RETURN, DEF FN, SUB..END SUB

1. Membuat subprogram dengan GOSUB..RETURN

   Perintah GOSUB..RETURN merupakan salah satu cara pemrograman moduler yang paling sederhana. Perintah ini sebenarnya agak mirip dengan perintah GOTO.

   Perbedaannya adalah pada perintah GOTO seteleh loncat ke suatu baris tidak ada kewajiban untuk kembali ke baris/perintah sebelumnya, sedangkan pada perintah GOSUB..RETURN pasti kembali ke tempat semula.

   Bentuk umum dari instruksi ini adalah :

```
GOSUB [nomor baris / label]
    ...
    ...
nomor baris / label :
    ...
    ...
RETURN
```

Contoh :

```
CLS
Print "Percobaan perintah GOSUB..RETURN"
GOSUB gambargaris
Print "Belajar Bahasa BASIC"
GOSUB gambargaris
END
gambargaris :
   print "********************"
RETURN
```

Output :

```
Percobaan perintah GOSUB..RETURN
********************
Belajar Bahasa BASIC
********************
```

Contoh 2 :

```
CLS
REM Program menghitung akar dengan rumus ABC
REM diselesaikan menggunakan perintah GOSUB
Input a,b,c
D=b^2-4*a*c
IF D>0 then Gosub akarreal
IF D=0 then Gosub akarsama
IF D<0 then gosub akarimajiner
END
akarreal :
  x1=(-b+sqr(d))/(2*a)
  x2=(-b-sqr(d))/(2*a)
  print "Ada 2 akar real yaitu "; x1;" dan ";x2
RETURN

akarsama :
  x1=-b/(2*a)
  print "akarnya kembar yaitu :";x1
RETURN

akarimajiner :
  nyata = -b/(2*a)
  imajiner = sqr(-d)/(2*a)
  print "Akarnya imajiner yaitu : "
  print "Akar 1 : "; nyata;"+";imajiner;"i"
  print "Akar 2 : "; nyata;"-";imajiner;"i"
RETURN
```

2. Membuat fungsi dengan perintah DEF..FN

   Perintah DEF FN adalah perintah yang digunakan untuk membuat fungsi sendiri. Jika fungsi yang akan dibuat terdiri atas satu baris maka format penulisannya cukup seperti berikut ini :

   DEF Fnnamafungsi(parameter)=fungsi

   Jika fungsi berisi lebih dari 1 baris, maka format penulisannya adalah :

   DEF Fnnamafungsi(parameter)
   ...
   blok instruksi
   ...
   Fnnamafungsi=....(nilai dari fungsi)
   END DEF

   Contoh :

```
REM Contoh program menggunakan fungsi
DEF FNdiskrim(a,b,c)=b^2-4*a*c
Input "Masukkan nilai a,b,c : ",a,b,c
? FNdiskrim(a,b,c)
? FNdiskrim(2,8,4)
x=10
y=20
z=5
? FNdiskrim(x,y,z)
```

   Output :
   Masukkan nilai a,b,c : 1,2,3
   -8 (hasil dari nilai yang diinputkan a=1, b=2 dan c=3)
   32 (hasil dari pemanggilan fungsi FNdiskrim(2,8,4)
   200 (hasil dari x=10, y=20, z=5)

Pada contoh program di atas Fungsi Fndiskrim mempunyai 3 parameter yaitu variabel a,b dan c. Untuk memanggil fungsi, variabel yang digunakan boleh tidak sama seperti pada contoh di atas Fndiskrim(x,y,z) artinya nilai variabel x,y dan z akan diproses oleh fungsi sebagai variabel a, b dan c.

```
DEF FNdiskrim(a,b,c)=b^2-4*a*c
a=10
b=20
c=30
? FNdiskrim(a,b,c)
? FNdiskrim(b,c,a)
? FNdiskrim(c,a,b)
```

Output :
-800 (a=10, b=20, c=30)
100 (a=20, b=30, c=10)
-2300 (a=30, b=10, c=20)

Pada contoh di atas terlihat bahwa urutan dari parameter sangat mempengaruhi perhitungan nilai dari fungsi.

Contoh berikut ini akan menunjukkan betapa bergunanya pemaiakan fungsi pada banyak kasus pemrograman.

Banyak kombinasi dapat dihitung dengan rumus :

$$Cn = N! / ((N-R)! . R!)$$

N= banyaknya data yang akan dikombinasikan

R = banyaknya kombinasi

C = jumlah kombinasi yang terjadi

Misalnya jumlah dari data yang akan dikombinasikan adalah sejumlah 3 buah data yaitu A,B dan C. Ketiga data tersebut akan dikombinasikan dua-dua (R=2), maka kombinasi yang akan terjadi adalah : AB, AC dan BC (3 macam). Rumus untuk menghitung jumlah kombinasi menggunakan faktorial.

```
DEF FNfakt(X)
LOCAL i,kali
  kali=1
  FOR i=1 to X
     kali=kali*i
  NEXT
  FNfakt=kali
END DEF

CLS
input "Masukkan banyaknya data   : ",N
input "Masukkan banyak kombinasi : ",R
C = FNfakt(N)/(FNfakt(N-R)*FNfakt(R))
? "Jumlah kombinasi yang terjadi : ",C
```

Output :

```
Masukkan banyaknya data   : 5
Masukkan banyak kombinasi : 3
Jumlah kombinasi yang terjadi :10
```

Pada contoh program di atas, fungsi Fnfakt dipanggil sebanyak 3 kali dengan menggunakan parameter yang berbeda.

```
C = FNfakt(N)/(FNfakt(N-R)*FNfakt(R))
```

Pemanggilan fungsi yang pertama menggunakan N sebagai variabel yang dimasukkan ke parameter X, kemudian (N-R) dan yang terakhir menggunakan variabel R. Dengan menggunakan fungsi maka program menjadi lebih ringkas dan efisien. Bandingkan jika untuk menyelesaikan kasus seperti pada contoh program di atas tidak digunakan fungsi, maka pasti programnya menjadi jauh lebih panjang karena bagian untuk menghitung faktorial harus ditulis berulang kali.

Penggunaan statement **LOCAL i,kali** artinya variabel I dan kali adalah variabel lokal yang hanya dikenali di dalam fungsi Fnfakt dan tidak dikenali di bagian program yang lain.

Berikut ini adalah suatu program dengan fungsi untuk menghitung a pangkat b. A dan b berupa parameter.

```
DEF FNhitung(a,b)
LOCAL i,hasil
   hasil=1
   FOR i=1 to b
      hasil = hasil * a
   NEXT
   FNhitung=hasil
END DEF

CLS
x=4 : y=3
? FNhitung(x,y)
? FNhitung(2,5)
? FNhitung(3,3)
```

Output :

64 ---→ diperoleh dari 4 pangkat 3
32 ---→ diperoleh dari 2 pangkat 5
27 ---→ diperoleh dari 3 pangkat 3

Fungsi dan parameter dapat bertipe numerik maupun string tergantung dari kasus yang sedang dipecahkan. Berikut ini adalah contoh fungsi yang bertipe string. Fungsi tersebut berguna untuk membalik kata.

```
DEF FNbalikkata$(kata$)
LOCAL i,kata2$
  kata2$=""
  FOR i=len(kata$) to 1 step -1
     kata2$ = kata2$+ mid$(kata$,i,1)
  next
  FNbalikkata$ = kata2$
END DEF

REM Program Utama
CLS
INPUT "Masukkan sembarang kata : ",kata$
? "Kata setelah dibalik : "; FNbalikkata$(kata$)
```

Output :
Masukkan sembarang kata : Surabaya Kota Pahlawan
Kata setelah dibalik : nawalhaP atoK ayabaruS

3. Membuat procedure dengan perintah SUB..END SUB

Pada bahasa BASIC untuk membuat subroutine yang berupa procedure adalah dengan perintah SUB..END SUB.

Bentuk umum perintah SUB..END SUB adalah sebagai berikut :

```
SUB nama [daftar parameter]
        …
        kumpulan instruksi
        …
END SUB
```

Untuk memanggil subroutine / procedure digunakan perintah :

```
CALL nama_procedure
```

Dibandingkan dengan perintah GOSUB..RETURN perintah ini mempunyai beberapa keunggulan yaitu :

- Mempunyai parameter
- Cara pemanggilannya lebih mudah
- Variabel yang digunakan dapat bersifat global maupun local
- Dapat dipanggil secara recursif (memanggil dirinya sendiri)

Seperti halnya pada FUNCTION, pada procedure juga terdapat 2 jenis variabel yaitu variabel LOCAL dan variabel GLOBAL / SHARED. Variabel local artinya variabel tersebut hanya dikenali oleh procedure itu saja dan tidak dikenali oleh bagian program yang lain. Sedangkan variabel bertipe global / shared adalah variabel yang dikenali oleh semua bagian program.

Contoh :

```
SUB diskriminan(a,b,c)
LOCAL D
  D=b^2-4*a*c
  ?D
END SUB

CLS
CALL diskriminan(10,30,5)
CALL diskriminan(20,5,10)
CALL diskriminan(10,25,5)
```

Output :

```
700
-775
425
```

Jika program di atas dimodifikasi sedikit menjadi sebagai berikut.

```
SUB diskriminan(a,b,c)
LOCAL D
  D=b^2-4*a*c
END SUB

CLS
CALL diskriminan(10,30,5)
?D
CALL diskriminan(20,5,10)
?D
CALL diskriminan(10,25,5)
?D
```

Output :

```
0
0
0
```

Semua pemanggilan procedure menghasilkan output 0 karena variabel D dibuat bertipe lokal sehingga hanya dikenali oleh procedure diskriminan dan tidak dikenali oleh program utama. Oleh karena itu, pada saat variabel D dicetak di dalam program utama hasilnya selalu 0. Berbeda dengan program versi sebelumnya, variabel D dicetak di dalam procedure diskriminan sehingga hasilnya sesuai dengan perhitungan.

Agar hasilnya tetap benar, maka pada procedure diskriminan variabel D harus dibuat bertipe SHARED seperti contoh berikut ini :

```
SUB diskriminan(a,b,c)
SHARED D    ----------------> Variabel D diubah dari bertipe LOCAL
  D=b^2-4*a*c                   menjadi bertipe SHARED
END SUB
```

Output :

```
700
-775
425
```

```
SUB faktorial(X,hasil)
Local i
hasil=1
For i= 1 to X
    hasil=hasil*i
next
END SUB

CLS
input "Masukkan banyaknya data   : ",N
input "Masukkan banyak kombinasi : ",R
CALL faktorial(N,hasil1)
CALL faktorial(N-R,hasil2)
CALL faktorial(R,hasil3)
C = hasil1 / (hasil2 * hasil3)
? "Jumlah kombinasi yang terjadi : ",C
```

Output :

```
Masukkan banyaknya data   : 5
Masukkan banyak kombinasi : 3
Jumlah kombinasi yang terjadi :10
```