

PEMROGRAMAN MODULAR



Overview

Pemrograman modular memungkinkan perancang program menyederhanakan persoalan didalam program dengan memecah atau membagi persoalan tersebut menjadi sub-sub persoalan yang lebih kecil agar mudah diselesaikan. Secara umum dikenal dua cara yang dapat digunakan untuk memecah persoalan dalam modul-modul, yaitu dengan menggunakan struktur fungsi dan prosedur. Pemahaman tentang perbedaan dan karakteristik masing-masing struktur tersebut perlu diimbangi pula dengan kemampuan mengimplementasikannya dalam program.



Tujuan

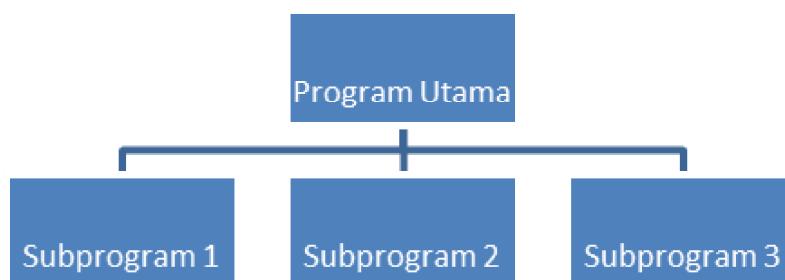
- Memahami konsep pemrograman modular
- Mengetahui dua cara pemrograman modular: fungsi dan prosedur
- Mengetahui cara mengimplementasikan fungsi dan prosedur dalam program
- Mengetahui dan dapat menerapkan pemanggilan subprogram dari program utama.

10.1 Definisi Pemrograman Modular

Dalam sebuah program, sering kali pemrogram perlu memecah persoalan yang kompleks menjadi beberapa bagian yang lebih mudah diselesaikan. Ide inilah yang mencetuskan struktur pemrograman modular, yaitu memecah persoalan menjadi sub-sub persoalan yang biasa disebut subprogram.

Bayangkan sebuah program yang dibuat untuk menghitung nilai rata-rata dari sekumpulan nilai integer. Dalam prosesnya, program melakukan perhitungan tersebut dalam dua langkah, yaitu menjumlahkan seluruh nilai, kemudian membaginya dengan banyaknya nilai yang tersedia. Dengan demikian program tersebut dapat dipecah menjadi dua subprogram, yaitu subprogram penjumlahan dan subprogram pembagian.

Selain itu, pemrograman modular memungkinkan pemrogram memanggil kembali subprogram yang telah didefinisikannya setiap kali diperlukan dalam program tersebut. Pemrogram tidak perlu berulang kali mendefinisikan sekumpulan instruksi yang diperlukan beberapa kali dalam sebuah program maupun dalam program lainnya. Dengan pemrograman modular, sebuah subprogram dapat dianggap sebagai program kecil dengan sebuah tujuan spesifik yang umumnya berisi operasi sederhana dan apabila terdapat kesalahan dapat dilokalisir pada subprogram itu sendiri. Sub-sub program tersebut kemudian disatukan oleh bagian program utama yang dapat memanggil subprogram tersebut sesuai kebutuhan dalam program.



Gambar 10.1 Ilustrasi pemrograman modular

Dalam pemrograman, dikenal dua tipe subprogram yang biasa digunakan untuk memecah persoalan kompleks menjadi lebih sederhana, yaitu fungsi (*function*) dan prosedur (*procedure*). Kedua

tipe subprogram ini dapat digunakan bersamaan maupun salah satunya saja dalam sebuah program. Masing-masing tipe subprogram memiliki karakteristik dan perilaku yang berbeda sehingga penggunaannya dalam program juga berbeda-beda.

Subprogram sebagai bagian dari program utama wajib mendefinisikan kondisi awal (*initial state/I.S.*) sebelum proses dalam subprogram dieksekusi dan juga mendefinisikan kondisi akhir (*final state/F.S.*) yang berupa hasil proses (*output*) atau perubahan nilai dalam variabel tertentu (khusus untuk fungsi saja).

Beberapa fungsi dan prosedur telah terdefinisi dan dapat langsung digunakan oleh pemrogram dalam sebuah program dengan mendefinisikan variabel-variabel yang diperlukan. Selain fungsi dan prosedur yang telah terdefinisi tersebut, pemrogram juga dapat membuat sendiri fungsi dan prosedur yang diperlukannya dalam sebuah program.

Dalam membuat sebuah subprogram, pemrogram dapat menyimpannya dalam salah satu dari dua lokasi berikut ini:

- ↳ dalam file yang sama dengan program utama: dapat dilakukan jika subprogram sedikit dan berukuran kecil sehingga relatif mudah dikelola dalam sebuah file
- ↳ dalam file yang terpisah: biasanya dilakukan jika subprogram sudah terlalu banyak sehingga sulit dikelola, atau jika pemrogram menginginkan supaya subprogram dapat digunakan di beberapa program utama sekaligus

10.2 Variabel Lokal dan Variabel Global

10.2.1 Variabel Lokal

Dalam mendeklarasikan sebuah fungsi/ prosedur, dapat dideklarasikan pula variabel-variabel yang akan digunakan dalam fungsi/ prosedur tersebut. Variabel semacam ini disebut **variabel lokal** atau **variabel internal**, artinya variabel ini hanya dikenali secara lokal dalam sebuah subprogram (fungsi atau prosedur). Variabel lokal tidak dapat dipanggil, diakses dan diubah oleh prosedur atau fungsi yang lain, bahkan oleh program utama sekalipun

karena hanya dapat dikenali oleh prosedur atau fungsi dimana variabel ini didefinisikan.

10.2.2 Variabel Global

Sedangkan variabel yang didefinisikan dalam program utama dan dapat digunakan di program utama maupun sub-sub program lainnya disebut dengan **variabel global**. Nilai dari variabel ini dapat dipanggil, diakses dan diubah oleh prosedur atau fungsi apapun yang terdapat dalam program tersebut.

10.3 Fungsi

Fungsi adalah subprogram yang menerima data masukan, melakukan beberapa perhitungan dari data tersebut, kemudian mengembalikan output berupa sebuah data baru. Dengan kata lain, sebuah fungsi memetakan sebuah nilai (dalam *domain*) menjadi nilai lain (dalam *range*) dengan operasi/ proses tertentu. Pendeklarasian fungsi merupakan salah satu cara memecah persoalan ke dalam beberapa sub persoalan yang lebih mudah diselesaikan.

Dalam pembuatan sebuah fungsi, pemrogram harus mendefinisikan:

- ↳ nama fungsi
- ↳ Tipe data yang dibuat/ dihasilkan oleh fungsi
- ↳ Daftar parameter yang menyatakan data yang diperlukan oleh fungsi
- ↳ Satu atau lebih instruksi yang melakukan perhitungan

Selanjutnya, fungsi yang sudah didefinisikan dapat digunakan dalam program utama maupun dalam fungsi lainnya dengan cara memanggil nama fungsi dan memberikan parameter yang diperlukan oleh fungsi tersebut.

Fungsi bekerja menurut mekanisme pemanggilan-pengembalian (*call-return mechanism*). Tahapan dalam mekanisme tersebut adalah:

- ↳ Fungsi dipanggil dari program utama maupun fungsi lainnya
- ↳ Sekumpulan operasi dalam fungsi dieksekusi
- ↳ Hasil eksekusi dikembalikan ke program utama atau fungsi lain yang memanggilnya.

Pada intinya fungsi berguna untuk :

- a. Mengurangi pengulangan penulisan program yang berulangan atau sama.
- b. Program menjadi terstruktur, sehingga mudah dipahami dan dikembangkan.

Fungsi-fungsi yang sudah kita kenal sebelumnya adalah fungsi *main()*, yang bersifat mutlak, karena fungsi ini program akan dimulai, sebagai contoh yang lainnya fungsi *printf()* yang mempunyai tugas untuk menampilkan informasi atau data kelayar dan masih banyak lainnya.

10.3.1 Struktur Fungsi

Sebuah fungsi sederhana mempunyai bentuk penulisan sebagai berikut :

```
nama_fungsi(argumen)
{
    ...
    pernyataan / perintah;
    ...
    pernyataan / perintah;
```

Keterangan:

- ↳ Nama fungsi, boleh dituliskan secara bebas dengan ketentuan, tidak menggunakan spasi dan nama-nama fungsi yang mempunyai arti sendiri.
- ↳ Argumen, diletakan diantara tanda kurung “()” yang terletak dibelakang nama fungsi. Argumen boleh diisi dengan suatu data atau dibiarkan kosong.
- ↳ Pernyataan / perintah, diletakan diantara tanda kurung ‘{ }’.

Contoh :

Contoh definisi kuadrat() :

```
// Prototipe fungsi  
long kuadrat (long l);
```

```
-----  
// Definisi fungsi  
long kuadrat(long l)  
{  
    return(l * l);  
}
```

Pernyataan **return** di dalam fungsi digunakan untuk memberikan nilai balik fungsi. Pada contoh diatas, fungsi kuadrat() memberikan nilai balik berupa nilai kuadrat dari argumen.

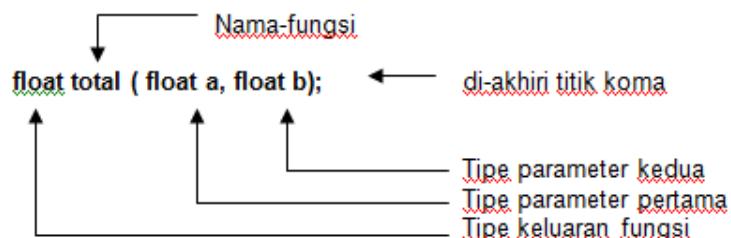
10.3.2 Prototipe Fungsi

Prototipe fungsi digunakan untuk menjelaskan kepada kompiler mengenai :

- ↳ Tipe keluaran fungsi.
- ↳ Jumlah parameter.
- ↳ Tipe dari masing-masing parameter.

Salah satu keuntungan pemakai prototipe, kompiler akan melakukan konversi antara tipe parameter dalam definisi dan parameter saat pemanggilan fungsi tidak sama atau akan menunjukkan kesalahan jika jumlah parameter dalam definisi dan saat pemanggilan berbeda.

Contoh prototipe fungsi :



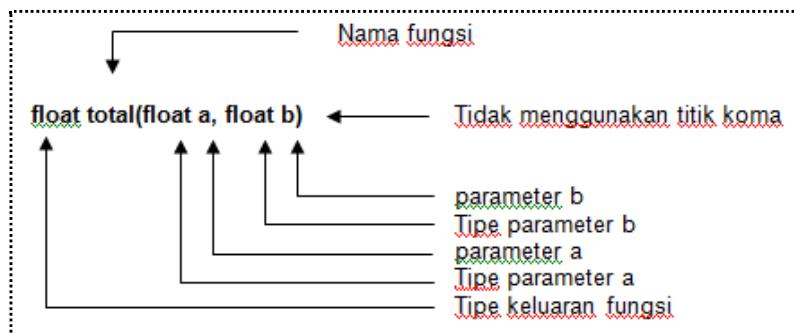
Jika dalam penggunaan fungsi yang dideklarasikan dengan menggunakan prototipe, maka bentuk definisi harus diubah. Sebagai contoh pada pendefinisian berikut :

```

float total(a, b)
float a, y;

```

Bentuk pendefinisian diatas harus diubah menjadi bentuk modern pendefinisian fungsi :

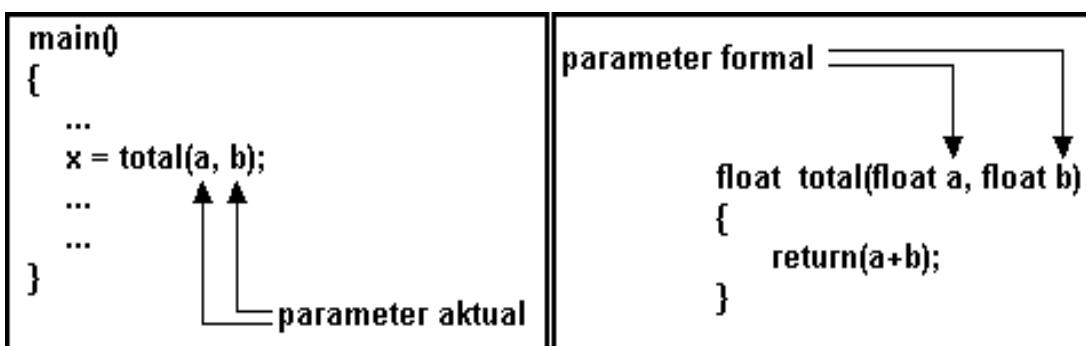


10.3.3 Parameter Fungsi

Terdapat dua macam para parameter fungsi, yaitu :

- ↳ **Parameter formal** adalah variabel yang ada pada daftar parameter dalam definisi fungsi.
- ↳ **Parameter Aktual** adalah variabel yang dipakai dalam pemanggilan fungsi.

Bentuk penulisan Parameter Formal dan Parameter Aktual.



Ada dua cara untuk melewaskan parameter ke dalam fungsi, yaitu berupa :

- ↳ Pemanggilan dengan nilai (*Call by Value*)
- ↳ Pemanggilan dengan Referensi (*Call by Reference*)

10.3.4 Pemanggilan dengan nilai (*Call by Value*)

Pemanggilan dengan nilai merupakan cara yang dipakai untuk seluruh fungsi buatan yang telah dibahas didepan. Pada pemanggilan dengan nilai, nilai dari parameter aktual akan ditulis keparameter formal. Dengan cara ini nilai parameter aktual tidak bisa berubah, walaupun nilai parameter formal berubah.

```
/* ----- */
/* Penggunaan Call By Value */
/* Program Pertukaran Nilai */
/* ----- */

#include<conio.h>
#include<stdio.h>
#include<iostream.h>

tukar(int x, int y);

main()
{
    int a, b;

    a = 88;
    b = 77;

    clrscr();

    cout<<"Nilai Sebelum Pemanggilan Fungsi";
    cout<<"\na = "<<a<<" b = "<<b;

    tukar(a,b);

    cout<<"\nNilai Setelah Pemanggilan Fungsi";
    cout<<"\na = "<<a<<" b = "<<b;

    getch();
}

tukar(int x, int y)
{
    int z;

    z = x;
    x = y;
    y = z;

    cout<<"\n\nNilai di dalam Fungsi Tukar()";
    cout<<"\nx = "<<x<<" y = "<<y;
    cout<<endl;
}
```

10.3.5 Pemanggilan dengan Referensi (*Call by Reference*)

Pemanggilan dengan reference merupakan upaya untuk melewaskan alamat dari suatu variabel kedalam fungsi. Cara ini dapat dipakai untuk mengubah isi suatu variabel diluar fungsi dengan melaksanakan pengubahan dilakukan didalam fungsi.

```
/* ----- */
/* Penggunaan Call By Reference */
/* Program Pertukaran Nilai */
/* ----- */

#include<conio.h>
#include<stdio.h>
#include<iostream.h>

tukar(int *x, int *y);

main()
{
    int a, b;
    a = 88;
    b = 77;

    clrscr();

    cout<<"Nilai Sebelum Pemanggilan Fungsi";
    cout<<"\na = "<<a<<" b = "<<b;

    tukar(&a,&b);

    cout<<endl;
    cout<<"\nNilai Setelah Pemanggilan Fungsi";
    cout<<"\na = "<<a<<" b = "<<b;

    getch();
}

tukar(int *x, int *y )
{
    int z;

    z = *x;
    *x = *y;
    *y = z;

    cout<<endl;
    cout<<"\nNilai di Akhir Fungsi Tukar()";
    cout<<"\nx = "<<*x<<" y = "<<*y;
}
```

Contoh 32	Hasil
<pre>/* /* Contoh 32 : Pembuatan fungsi dengan argumen * /* bertipe long dan nilai balik berupa * /* long * */ #include <iostream.h> #include <iomanip.h> #include <conio.h> long kuadrat(long1); // prototipe fungsi void main() { clrscr(); for (long bil = 200; bil < 2000; bil+= 200) cout << setw(8) << bil << setw(8) << kuadrat(bil) << endl; } // Definisi fungsi long kuadrat(long 1) { return(1 * 1); }</pre>	200 40000 400 160000 600 360000 800 640000 1000 1000000 1200 1440000 1400 1960000 1600 2560000 1800 3240000

Contoh 33	Hasil
<pre>/* /* Contoh 33 : Menggambarkan nilai bawaan dalam * /* argumen fungsi */ #include <iostream.h> #include <conio.h> void tulis_cplus(int jum); // Prototipe fungsi void main() { clrscr(); tulis_cplus(1); // Untuk menuliskan sebuah tulisan C++ } void tulis_cplus(int jum); { for (int i = 0; i < jum; i++) cout << " C++ " << endl; cout << " Selesai " << endl; }</pre>	C++ Selesai

Contoh 34	Hasil
<pre data-bbox="223 350 794 977"> /* /* Contoh 34 : Menggambarkan nilai bawaan * /* Dalam argumen fungsi * /* #include <iostream.h> #include <conio.h> void tulis_cplus(int jum = 1); // Prototipe fungsi // Dan menyetel nilai bawaan fungsi void main() { clrscr(); tulis_cplus(); // Argumen tidak perlu disebutkan } void tulis_cplus(int jum); { for (int i = 0; i < jum; i++) cout << " C++ " << endl; cout << " Selesai " << endl; } </pre>	C++ Selesai

Pada contoh program 33 dan 34 mempunyai kesamaan hanya saja pada contoh program 34 dalam prototipe fungsi nilai bawaannya dikutsertakan sehingga pada saat argumen pemanggilan fungsi tidak perlu di tuliskan lagi.

10.4 Prosedur

Cara lain memecah persoalan pemrograman ke dalam sub-sub persoalan pemrograman adalah dengan mendeklarasikan prosedur. Prosedur adalah sederetan instruksi yang diberi nama, dan melakukan tujuan tertentu. Seperti halnya pada fungsi, prosedur bekerja dengan mekanisme pemanggilan-pengembalian (*call-return mechanism*), yaitu dengan urutan langkah:

- ↳ Prosedur dipanggil oleh kode pemanggil (program utama maupun prosedur lainnya)
- ↳ Sekumpulan operasi yang disimpan dalam prosedur dieksekusi
- ↳ Kontrol dikembalikan ke kode pemanggil

Bentuk umum :

```
Procedure Nama_Prosedur(param1:tipedata,param2:tipedata,)
```

Contoh:

- ↳ Procedure TambahKali; (procedure tanpa parameter).
- ↳ Procedure Hitung(a,b : integer); (procedure dengan parameter).

Dengan catatan bahwa nama prosedur dan nama parameternya harus disebutkan dalam blok kode pemanggil. Berbeda dengan fungsi, daftar parameter pada procedure terbagi menjadi dua yaitu parameter input dan parameter output. Daftar parameter boleh kosong (tidak ada parameter input maupun output). Jika parameter tidak kosong (minimal ada satu parameter) maka harus dituliskan nama parameter beserta tipe datanya.

Prosedur tanpa parameter memanfaatkan nilai dari variabel yang terdefinisi dalam kode program utama/prosedur lain yang memanggilnya. Prosedur tanpa parameter ini hanya dapat dieksekusi jika nilai dari variabel yang diperlukan dalam prosedur sudah didefinisikan dalam kode program utama/ prosedur lain yang memanggilnya.

Prosedur dengan parameter dibuat untuk mengeksekusi sekumpulan instruksi dengan parameter yang berbeda-beda. Nama parameter yang dituliskan pada definisi / spesifikasi prosedur disebut dengan **parameter formal**. Sedangkan parameter yang dituliskan pada pemanggilan prosedur disebut **parameter aktual**.

Parameter formal adalah nama-nama variabel yang dipakai dalam mendefinisikan prosedur, dan membuat prosedur tersebut dapat dieksekusi dengan variabel yang berbeda saat pemanggilan. Terdapat tiga tipe parameter formal:

- ↳ parameter input, yaitu parameter yang diperlukan prosedur sebagai masukan untuk melakukan aksi yang efektif
- ↳ parameter output, yaitu parameter yang akan menyimpan nilai yang dihasilkan oleh prosedur
- ↳ parameter input/output, yaitu parameter yang diperlukan prosedur sebagai masukan untuk melakukan aksi tertentu, yang

pada akhir prosedur akan diisi dengan nilai baru sebagai hasil eksekusi prosedur

Parameter aktual adalah variabel / konstanta yang dipakai ketika prosedur dipanggil oleh program utama / prosedur lain. Parameter aktual dapat berupa variabel / konstanta, tapi parameter output harus berupa variabel karena akan menyimpan hasil eksekusi prosedur.

Struktur pemanggilan prosedur dari program utama maupun prosedur lain adalah hanya dengan menuliskan nama procedurenya kemudian diikuti daftar parameternya sebagai berikut:

nama_prosedure

Sama halnya dengan fungsi, prosedur dapat terhubung dengan program utama maupun prosedur lain melalui pertukaran parameter. Bedanya, deklarasi prosedur harus menyertakan nama dan tipe data dari seluruh parameter input dan outputnya.

Untuk dapat menggunakan prosedur tersebut perlu adanya pemanggilan prosedur sama seperti pemanggilan fungsi. Untuk lebih jelasnya kita lihat potongan program berikut :

```
void hitungluaslingkaran()
{
    float phi;
    float r;
    luas=phi*r*r;
    printf("luas lingkaran : %0.2f\n",luas);
}
```

Pada prosedur di atas kita mencari luas lingkaran dengan menggunakan rumus πr^2 dengan deklarasi lokal berupa phi dan r.

Pola pemanggilan *procedure* juga mengikuti aturan yang diterapkan pada *function*. Dari algoritma di atas, dapat dituliskan ke dalam bahasa pemrograman C++ sebagai berikut:

1. #include <stdio.h>
2. //Variabel Global
3. float nilai1, nilai2, rataan;
- 4.

```

5.  {==Program Utama==}
6.  void main () { // program Utama
7.      // Inisialisasi dua buah nilai
8.      nilai1 = 8;
9.      nilai2 = 16;
10.
11.     /* pemanggilan prosedur1 */
12.     hitung_rataan(nilai1,nilai2,rataan);
13.     /* menampilkan nilai rata-rata */
14.     cout rataan;
15. }
16.
17. {==Prosedur2 hitung_jumlah==}
18. void hitung_jumlah(int pertama, int kedua, int& jumlah)
19. {
20.     jumlah = pertama + kedua;
21. }
22.
23. {==Prosedur1 hitung_rataan==}
24. void hitung_rataan(int var1, int var2,int& rata)
25. {
26.     int jml;
27.     // panggil procedure 2
28.     hitung_jumlah(var1,var2,jml);
29.     rata = jml / 2;
30. }

```

Perbedaan utama yang terlihat antara fungsi dan prosedur adalah bahwa prosedur tidak perlu mengembalikan sebuah nilai, sedangkan fungsi harus selalu mengembalikan nilai (ditunjukkan dengan perintah return ... di akhir blok fungsi). Selain itu, kode pemanggil fungsi perlu mendefinisikan sebuah variabel untuk menyimpan nilai yang dikembalikan oleh fungsi, sedangkan pada prosedur tidak demikian. Pengisian variabel dilakukan oleh prosedur sehingga kode pemanggil tidak perlu lagi mempersiapkan sebuah variabel penyimpan hasil eksekusi prosedur.

Pada procedure pertama dan kedua terdapat lambang “&” setelah penulisan tipe data pada parameter procedure. Hal itu maksudnya adalah variabel tersebut merupakan parameter input/output, dengan kata lain akan terjadi pemetaan dua arah antara

variabel pada parameter procedure dengan variabel pada parameter pemanggilan procedure.

10.5 Fungsi dan Prosedur yang telah terdefinisi

Selain dapat membuat sendiri fungsi atau prosedur yang diperlukan dalam sebuah program, bahasa pemrograman juga sudah menyediakan beberapa fungsi dan prosedur yang sudah terdefinisi dan dapat langsung digunakan / dipanggil dalam program. Penggunaan fungsi maupun prosedur yang telah terdefinisi tersebut dapat mempermudah perancang program menyelesaikan sub persoalan tertentu.

Beberapa fungsi yang telah terdefinisi, antara lain:

FUNGSI	CONTOH
- Fungsi <i>ceil</i> (untuk membulatkan keatas nilai pecahan).	var-int $\leftarrow \text{ceil}(\text{ekspresi float})$
- Fungsi <i>min/ max</i> (menentukan nilai minimal atau maksimal dari dua bilangan)	var-int $\leftarrow \text{min}(3,5)$ var-int $\leftarrow \text{max}(3,5)$
- Fungsi <i>random</i> (mendapatkan nilai secara acak dari rentang tertentu)	var-int $\leftarrow \text{random}(10)$
- Fungsi <i>sin</i> (memperoleh nilai sinus dari suatu bilangan)	Var-float $\leftarrow \text{sin}(\text{int})$

Beberapa prosedur yang telah terdefinisi, antara lain:

PROSEDUR	CONTOH
- Prosedur <i>assert</i> (mengecek error pada ekspresi boolean)	assert (eks-boolean [,string])
- Prosedur <i>arg</i> (mengembalikan argumen ke-i dari program dan meyimpannya dalam sebuah string)	arg (eks-integer, var-string)
- Prosedur <i>date</i> (menampilkan tanggal sekarang)	date (var-string)
- Fungsi <i>time</i> (menampilkan jam sekarang dengan format jj:mm:dd)	time (var-string)

10.6 Fungsi Rekursif

Fungsi dapat dipanggil oleh program utama dan juga oleh fungsi yang lain, selain kedua metode pemanggilan tersebut, fungsi dapat juga dipanggil oleh dirinya sendiri. Yang dimaksud disini adalah pemanggilan fungsi itu didalam fungsi itu sendiri, bukan pada fungsi yang lain. Fungsi yang melakukan pemanggilan terhadap dirinya sendiri disebut dengan fungsi rekursif.

Berikut adalah contoh program dalam program yang menerapkan metode rekursif untuk menghitung nilai faktorial dari suatu bilangan. Struktur umum deklarasi prosedur adalah sebagai berikut:

```
1. // Program Hitung_Faktorial
2. #include <stdio.h>
3.
4. int angka;
5. int hasil;
6.
7. {==Program Utama==}
8. void main () { // program Utama
9.     printf("Masukan Angka Batas Atas Faktorial :");
10.    scanf("%i",&angka);
11.    hasil = faktorial(angka);
12.    printf("Faktorial Dati %i adalah %i.", angka, hasil);
13. }
14.
15. int faktorial(int bil)
16. {
17.     if bil = 0 then
18.         return 1
19.     else
20.         return (bil * faktorial(bil - 1));
21. }
```

Perhatikan pada baris ke-20 kode diatas yang diberi tanda arsir. Kode pemanggilan fungsi faktorial tersebut berada pada bloknya sendiri dengan parameter yang diubah. Hal yang harus diperhatikan dalam pembuatan rekursif adalah fungsi tersebut harus berhenti dimana didalam fungsi tersebut harus ada pengkondisian bahwa fungsi harus berhenti. Pada contoh diatas, code untuk menghentikan fungsi tersebut ada pada baris ke 17 dan 18, dimana apabila inputan parameternya 0,

maka akan menghasilkan 1. Namun jika tidak, proses akan memanggil fungsi yaitu dirinya sendiri.

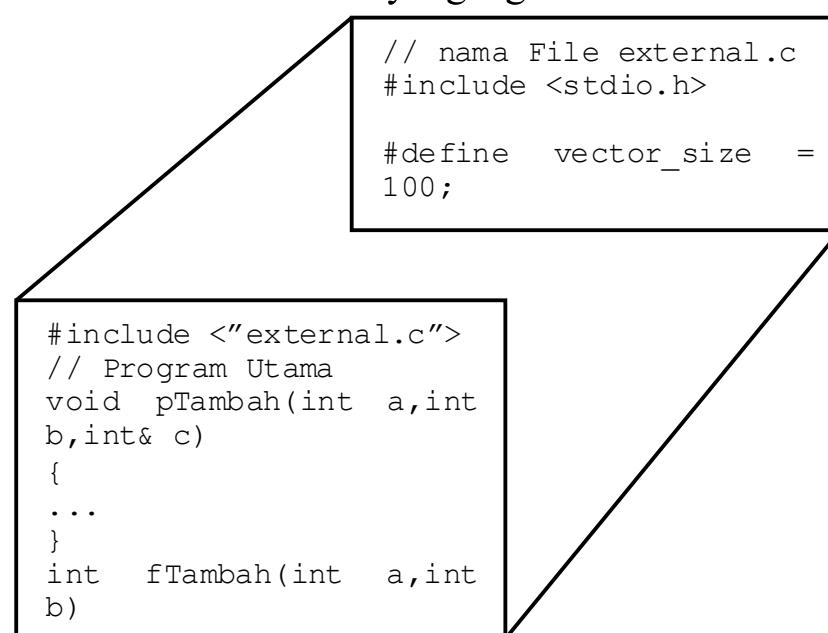
Sebagai ilustrasi program diatas, mari kita coba memanggil fungsi faktorial dengan Faktorial(5), maka proses yang akan dilakukan adalah :

```
1. 5 * Faktorial(4)
2. 5 * 4 * Faktorial(3)
3. 5 * 4 * 3 * Faktorial(2)
4. 5 * 4 * 3 * 2 * Faktorial(1)
5. 5 * 4 * 3 * 2 * 1 * Faktorial(0)
6. 5 * 4 * 3 * 2 * 1 * 1
7. 5 * 4 * 3 * 2 * 1
8. 5 * 4 * 3 * 2
9. 5 * 4 * 6
10. 5 * 24
11. 120
```

Hasil Akhir : 120

10.7 Unit

Fungsi dan prosedur dapat disimpan dalam file yang terpisah dari program utamanya. File-file semacam ini disebut sebagai unit. Fungsi, prosedur, dan variabel dapat disimpan dalam sebuah unit tanpa blok program utama. Pada saat dilakukan kompilasi, program utama akan mendeklarasikan unit-unit yang digunakan.



Gambar 10.1. Contoh deklarasi unit dalam program utama



Rangkuman

- Pemrograman modular adalah upaya memecah program yang kompleks ke dalam sub-subprogram yang lebih kecil untuk menyederhanakan penyelesaian persoalan.
- Setelah didefinisikan, subprogram dapat dipanggil berkali-kali dengan parameter yang berbeda-beda.
- Dua tipe subprogram yang biasa digunakan adalah fungsi (*function*) dan prosedur (*procedure*).
- Sebuah subprogram dapat disimpan dalam file yang sama dengan program utama, ataupun dalam file terpisah.
- Sebuah fungsi memetakan sebuah nilai (dalam *domain*) menjadi nilai lain (dalam *range*) dengan operasi / proses tertentu.
- Dalam penggunaan prosedur, dikenal dua tipe parameter, yaitu parameter formal dan parameter aktual.
- Fungsi dan prosedur bekerja menurut mekanisme pemanggilan-pengembalian (*call-return mechanism*).
- Terdapat beberapa fungsi dan prosedur yang telah terdefinisi dan dapat langsung digunakan dalam program.
- Fungsi, prosedur dan variabel yang disimpan dalam sebuah file yang terpisah dari program utamanya membentuk unit yang dapat digunakan oleh program utama.