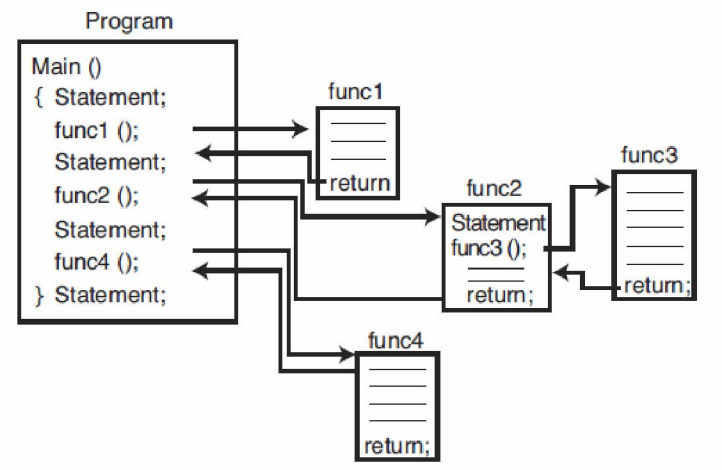
Chapter 4 Function

Fungsi

Fungsi (Function) adalah sekumpulan program yang diberi nama, sehingga dengan demikain jika program itu diperlukan dapat dipanggil kembali. Walaupun Pemrograman Berorientasi Objek telah menggeser perhatian dari fungsi ini, namun fungsi tetap saja merupakan bagian paling inti dalam suatu program. Fungsi global bisa berada di luar kelas maupun objek.

Fungsi dapat melakukan manipulasi terhadap data dan dapat mengembalikan suatu nilai. Semua program yang ditulis dengan bahasa C++ paling tidak mempunyai sebuah fungsi, yaitu main(), fungsi ini akan dipanggil secara otomatis ketika program dieksekusi, sedangkan fungsi yang lain baru akan bekerja ketia fungsi tersebut dipanggil. Karena fungsi ini bukan merupakan bagian dari objek, maka fungsi ini dipanggil secara global, dapat diakses dari manapun dalam program yang ditulis

Setiap fungsi diberi nama, dan ketika dalam suatu program dijumpai nama tersebut, maka eksekusi program akan dialihkan ke tubuh (isi) fungsi tersebut, setelah selesai, yaitu ditandai dengan statemen return atau tanda kurung kurawal tutup, maka akan kembali ke progam utama melanjutkan ke baris program berikutnya. Peristiwa ini dinamakan pemanggilan fungsi, berikut ini adalah ilustrasi mengenai pemanggilan fugsi :



Fungsi yang baik mengerjakan sebuah pekerjaan yang spesifik, mudah dipahami dan mudah dikenali berdasarkan nama fungsi tersebut. Pekejaan yang kompleks seharusnya dipecah-pecah menjadi beberapa fungsi yang nantinya dapat dipanggil ketika diperlukan.

Fungsi terdiri dari 2 macam, yaitu fungsi yang dibuat sendiri (*user-defined*) dan fungsi standard (*built-in*). Fungsi standard merupakan bagian dari paket kompiler yang kita pakai yang sudah tersedia untuk digunakan, sedangkan fungsi yang dibuat sendiri adalah fungsi yang kita tulis sebelum dapat dipergunakan.

Konsep Dasar Fungsi

Fungsi sebenarnya mirip dengan prosedur (pada bhs. Pascal), dan kedua hal ini disebut sebagai Subrutin**.** Kedua jenis subrutin ini (fungsi dan prosedur) memiliki kegunaan yang sama, yaitu melakukan tugas tertentu. Perbedaannya fungsi selalu mengembalikan suatu nilai setelah dipanggil sedangkan prosedur tidak.

Kita memerlukan subrutin, karena dalam program yang besar akan lebih baik jika tugas tertentu dilakukan oleh subrutin tertentu, dengan demikian program akan menjadi lebih mudah dibaca dan dipelihara.

Catatan :
Fungsi bisa dikatakan sebagai bentuk lain dari instruksi yang dapat memberikan sebuah nilai apabila diberi masukan yang dibutuhkan. Masukan tersebut dikenal dengan istiah Parameter.

Fungsi-fungsi merupakan elemen utama dari program bahasa C++. Program dari bahasa C++ dibentuk dari kumpulan fungsi, mulai dari fungsi utama dengan nama **main(),** fungsi-fungsi pustaka (standar) dan fungsi-fungsi yang dibuat sendiri oleh pemrogram (udf = *User Defined Function*). Fungsi-fungsi banyak digunakan dengan dua alasan utama, yaitu:

1. Fungsi-fungsi menjadikan program C++ mempunyai struktur yang jelas. Dengan memisahkan langkah–langkah detail ke satu atau lebih fungsi–fungsi, maka fungsi utama (**main()**) akan menjadi lebih pendek, jelas dan mudah dimengerti. Hal seperti ini menunjukan suatu struktur program yang baik.
2. Fungsi-fungsi dapat digunakan untuk menghindari penulisan program yang sama ditulis secara berulang-ulang. Selanjutnya bagian program yang membutuhkan langkah-langkah yang sama tidak perlu selalu dituliskan, melainkan cukup memanggil fungsi – fungsi tersebut.

Suatu fungsi harus diberi nama supaya dapat dipanggil dari bagian program yang membutuhkannya. Tugas yang dilakukan oleh suatu fungsi dapat berupa tugas input/output, penyeleksian atau tugas-tugas perhitungan dan sebagainya.

Mendefinisikan Fungsi

Secara umum, fungsi terdiri dari dua komponen yaitu definisi fungsi dan tubuh fungsi. Isi dari definisi fungsi adalah : tipe dari fungsi, nama dari suatu fungsi dan paramter-parameter yang digunakan. Tubuh dari fungsi berisikan statemen-statemen yang akan melakukan tugas yang diberikan oleh fungsi tersebut. Tubuh suatu fungsi diawali dengan tanda kurung kurawal buka dan diakhiri dengan tanda kurung kurawal tutup. Beikut ini adalah bentuk umum dari suatu fungsi:

```
<tipe> <nama_fungsi>([<paramter1>, <paramter2> ,...])
{
   <tubuh fungsi>
   [return <ekspresi>]
}
```

Definisi fungsi ditulis sebelum dituliskan tubuh fungsi dan tidak diakhiri dengan tanda titik koma. Tipe dari definisi fungsi sesuai dengan tipe data dari nilai yang dikembalikan jika fungsi itu mempunyai statment **return**, jika tidak terdapat statement **return** tipe ini diberi tipe **void**. Nama suatu fungsi dibentuk sendiri oleh pemrogram sesuai dengan syarat penamaan identifier yang telah dibahas pada bab 2 dan nama fungsi yang baik mencerminkan pekerjaan dari fungsi tersebut. Parameter suatu fungsi dapat dituliskan dengan dipisahkan oleh tanda koma, bisa mempunyai beberapa parameter namun dapat juga tidak mempunyai parameter sama sekali. Parameter dibutuhkan jika dalam tubuh fungsi memerlukan nilai dari luar fungsi. Parameter ini dinamakan parameter formal. Berikut ini adalah contoh cara mendefinisikan fungsi.

```
int terbesar(int bil1, int bil2)
{
    int hasil;
    if (bil1>bil2)
      kembali = bil1;
    else
      kembali = bil2;
      return kembali;
}
```

Deklarasi Fungsi (*Prototype*)

Suatu fungsi harus dideklarasikan sebelum digunakan, jika suatu fungsi tidak dideklarasikan maka fungsi tersebut tidak akan bisa dipanggil. Deklarasi tersebut akan memberitahukan kepada kompiler mengenai nama fungsi, tipe data kembalian dan parameter dari fungsi, sedangkan definisi dari fungsi memberitahukan kepada kompiler mengenai cara kerja fungsi. Deklarasi fungsi ini dinamakan prototipe (***prototype***).

Ada tiga cara mendeklarasikan fungsi (membuat *prototype*), yaitu :

- Menuliskan prototipe ke dalam sebuah file, kemudian menggunakan directive **`#include`** untuk menyertakannya.
- Tuliskan prototype di dalam file yang memakai fungsi tersebut.
- Definisikan fungsi di file yang memakai fungsi tersebut di posisi sebelum pemanggilnya, dengan demikian definisi fungsi ini bertidak sebagai prototype itu sendiri.

Meskipun kita dapat mendefinisikan fungsi sebelum digunakan, sehingga bisa menghindari pembuatan

prototype, namun cara ini merupakan cara yang tidak baik karena tiga alasan. Pertama, menampilkan fungsi dalam sebuah file dengan urutan tertentu adalah tidak baik, karena akan menyulitkan ketika terjadi perubahan program.

Kedua, ada kemungkinan fungsi pertama memerlukan pemanggilan fungsi kedua, tetapi ada juga kemungkinan fungsi kedua memanggil fungsi yang pertama. Pada kasus semacam ini tidak mungkin menempatkan definisi fungsi pada urutan yang benar tanpa membuat prototype.

Ketiga, penggunaan prototype merupakan teknik penelusuran kesalahan yang baik dan handal. Ketika suatu prototype mendeklarasikan fungsi dengan parameter tertentu dan nilai kembalian tertentu, maka kompiler akan menjaga konsistensinya dengan definisi fungsi tanpa harus menunggu program dijalankan.

Compiler C++ dapat memeriksa tipe data melalui parameter-parameter (actual parameter) yang dikirimkan dari program yang menggunakannya, dengan terlebih dahulu menyebutkan prototype fungsi tersebut. Jika terjadi kesalahan perbedaan antara tipe-tipe data parameter nyata yang dikirim dengan tipe-tipe data parameter formalnya, maka dapat diketahui melalui ketidakcocokan antara compiler untuk tipe data tersebut.

Prototype fungsi standard berada di file-file judulnya, dalam fungsi pustaka sebagai contoh, fungsi pustaka printf(), prototypenya berada di dalam file judul stdio.h. Pencantuman prototype fungsi dapat menggunakan preprocessor directive **#include**.

Labs.1 Membuat Fungsi yang mengembalikan nilai.

1. Buka Qt Creator dan buat project Qt Console Application baru dengan nama **Labs1**, kemudian tulis kode berikut.

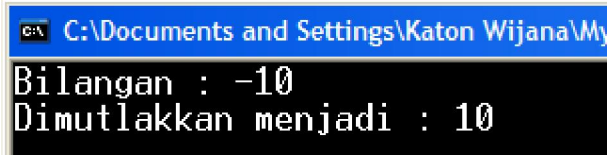
```
#include <QtCore/QCoreApplication>
#include <iostream>
int absolut(int bil);
int main(int argc, char *argv[])
{
    using namespace std;
    QCoreApplication a(argc, argv);
    int bilangan = -10;
    cout << "Bilangan : " << bilangan << endl;
    cout << "Dimutlakkan menjadi : " << absolut(bilangan) << endl;
    return a.exec();
}
int absolut(int bil){
    if(bil<0)
        return - bil;
    else
        return bil;
}
```

Prototype

Pemanggil fungsi

Definisi fungsi

2. Kemudian jalankan kode diatas dengan menekan tombol Ctrl+R, outputnya adalah sebagai berikut.



Analisa Program :

- Pada program diatas baris ketiga tertulis : `int absolut(int bil);` inilah yang disebut sebagai prototype, ditulis sebelum fungsi yang memakainya, yaitu main().
- Pada tubuh pogram, terdapat pemanggilan fungsi :
  `cout << "Dimutlakkan menjadi : " << absolut(bilangan) << endl;`
  Tampak pada hasil eksekusi bahwa nama fungsi tersebut digantikan dengan nilai **10**, yaitu nilai kembalian fungsi, ini menunjukkan bahwa fungsi **absolut**() mengembalikan nilai.
- Di bawah fungsi main() terdapat sebuah blok program dengan nama absolut(), inilah yang dinamakan definisi fungsi.

Catatan :

- Nama parameter pada prototype tidak harus sama dengan nama parameter pada definisi fungsi, oleh karena itu prototype tersebut di atas boleh juga dituliskan seperti berikut :

```
int absolut(int x);
```

- Nama parameter pada prototype tidak harus disebutkan, oleh karena itu prototype tersebut di atas boleh juga dituliskan seperti berikut :

```
int absolut(int);
```

Hasil Balik Fungsi

Suatu fungsi dalam menyelesaikan tugasnya, dapat hanya melakukan suatu tugas tanpa memberikan suatu nilai kembalian atau melakukan suatu tugas yang kemudian memberikan suatu nilai kembalian. Fungsi yang hanya menampilkan hasil di layar merupakan suatu fungsi yang hanya melakukan tugasnya saja tanpa memberikan hasil balik. Untuk membuat fungsi yang tidak mempunyai nilai kembalian digunakan tipe data **void** untuk tipe fungsi tersebut dan pada tubuh definisi fungsi tidak ada satmenet **return**.

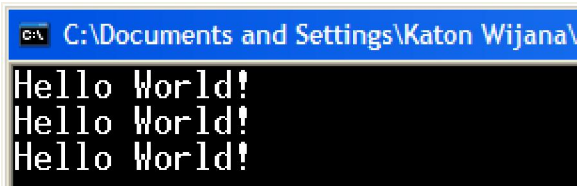
Labs.2 Membuat Fungsi yang tidak mengembalikan nilai.

1. Buka Qt Creator dan buat project Qt Console Application baru dengan nama **Labs2**, kemudian tulis kode berikut.

```
#include <QtCore/QCoreApplication>
```

```
#include <iostream>
void hello(int kali);
int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    hello(3);
    return a.exec();
}
void hello(int kali){
    using namespace std;
    for(int x=0;x<kali;x++)
        cout << "Hello World!" << endl;
}
```

2. Kemudian jalankan kode diatas dengan menekan tombol Ctrl+R, outputnya adalah sebagai berikut.



Analisa Program :

- Pada program diatas baris ketiga tertulis : `void hello(int kali);` tampak tipe dari fungsi ini adalah void, berarti tidak mengembalikan nilai.
- Pada tubuh pogram, terdapat pemanggilan fungsi :
  `hello(3);`
  Tampak pada hasil eksekusi bahwa nama fungsi ini dieksekusi bukan diakses nilainya (dicetak dengan **cout**), ini menunjukkan bahwa fungsi **hello()** tersebut tidak mengembalikan nilai.
- Di bawah fungsi main() terdapat sebuah blok program dengan nama hello(), pada tubuh fungsi ini tidak ada perintah return sama sekali, karena memang tidak mengembalikan nilai.

Jika suatu fungsi memberikan nilai kembalian, maka nilai kembalian yang diberikan oleh fungsi dapat dilakukan oleh statemen **return** yang diikuti oleh nilai hasil baliknya. Contoh fungsi yang mengembalikan nilai adalah seperti contoh pada Lab1 di atas.

Ruang Lingkup Variabel

Variabel-variabel memiliki ruang lingkup yang berbeda-beda, sesuai dengan ruang lingkup variabel, jenis-jenis variable dapat dibagi menjadi tiga:

Variable Lokal

Variable lokal merupakan variable yang hanya berlaku untuk pernyataan di dalam satu blok statemen saja, tidak dapat dipergunakan oleh blok lain, pendeklarasianya variabel lokal berada di dalam blok

statement (dalam kurung kurawal) yang bersangkutan. Variabel lokal akan dihapus dari memori jika proses sudah meninggalkan blok statemen letak variable lokalnya.

Labs.3 Variabel Lokal.

3. Buka Qt Creator dan buat project Qt Console Application baru dengan nama **Labs2**, kemudian tulis kode berikut.

```
#include <QtCore/QCoreApplication>
#include <iostream>
float kali(float a, float b);  /*prototype fungsi*/
int main(int argc, char *argv[])
{
    using namespace std;
    QCoreApplication a(argc, argv);
    float hasil;
    hasil = kali(4,7);
    cout << "Hasil = " << hasil << endl;
    return a.exec();
}

float kali(float a, float b)
{
        float c;
        c = a * b;
        return c;
}
```

4. Kemudian jalankan kode diatas dengan menekan tombol Ctrl+R, outputnya adalah sebagai berikut.

Analisa Program:

- Variable **a**, **b** dan **c** merupakan variabel lokal pada fungsi **kali**(). Variabel ini tidak dikenal pada fungsi utama sehingga variabel ini tidak dapat digunakan pada fungsi **main()** di atas, sebaliknya variabel **hasil** adalah variabel yang sifatnya lokal pada fungsi **main()**, sehingga tidak dikenal pada fungsi **kali()**.
- Jika variabel **a** atau **b** atau **c** dibaca pada fungsi **main()** maka akan terjadi kesalahan, yaitu bahwa variabel-variabel tersebut tidak dikenal (tidak dideklarasikan), demikian juga jika variabel hasil diakses di dalam fungsi **kali()**, maka variabel tersebut juga tidak akan dikenal.
- Variabel lokal sifat kerjanya hanya sekali. Jadi ketika fungsi **kali()** selesai dieksekusi, maka variabel **a**, **b** dan **c** dibebaskan dari memori, ketika fungsi ini dipanggil kembali di waktu lain, maka akan terjadi deklarasi (pemesanan tempat) lagi dan dianggap sebagai variabel baru.

Variable Global

Sesuai dengan namanya, variable global maksudnya adalah suatu variable yang dapat dikenali oleh

semua bagian dari program, tidak hanya terbatas pada satu blok statemen saja. Supaya menjadi variabel global, maka variabel global ini dideklarasikan di luar suatu blok ataupun di luar fungsi-fungsi yang mengguanakanya.

Labs.4 Variabel Global.

1. Buka Qt Creator dan buat project Qt Console Application baru dengan nama **Labs2**, kemudian tulis kode berikut.

```
#include <QtCore/QCoreApplication>
#include <iostream>
void kali(float a, float b);  /*prototype fungsi*/
float hasil; /*variabel global*/

int main(int argc, char *argv[])
{
    using namespace std;
    QCoreApplication a(argc, argv);
    kali(4,7);
    cout << "Variabel global hasil = " << hasil << endl;
    return a.exec();
}

void kali(float a, float b)
{
        hasil = a * b;
}
```

2. Kemudian jalankan kode diatas dengan menekan tombol Ctrl+R, outputnya adalah sebagai berikut.

Analisa Program:

- Variabel **hasil** dideklarasikan di luar blok program (di luar kurung kurawal), maka variabel **hasil** merupakan variabel global yang dikenal di blok manapun.
- Ketika variabel **hasil** mengalami manipulasi di dalam fungsi **kali()**, maka sebenarnya yang diubah adalah variabel **hasil** yang sama, sehingga ketika ditampilkan dengan **cout** variabel ini menghasilkan nilai perkalian antara **a** dan **b** seperti apa yang dilakukan di dalam fungsi kali().
- Perlu diperhatikan bahwa variabel **hasil** bersifat global bagi fungsi **main()** maupun fungsi **kali()** karena deklarasi variabel hasil tersebut diletakkan di atas kedua fungsi-fungsi tersebut. Jadi letak deklarasi suatu vaiabel yang diluar blok, menentukan cakupan sifat global variabel tersebut.

Variabel statik

Jika dilihat dari prinsip kerjanya, variabel statik bertentangan dengan variable lokal, variable lokal tidak lagi digunakan setelah suatu proses dalam blok selesai, namun variable static adalah jenis variabel yang masih tetap ada nilainya dan akan tetap dipertahankan nilainya walaupun sudah keluar dari proses. Sebenarnya variabel statik ini merupakan pengubah (*modifer*) dari variable lokal atau global, sehingga variabel statik dapat bersifat statik lokal atau statik global tergantung dari letak pendeklarasianya.

Labs.5 Variabel Statik.

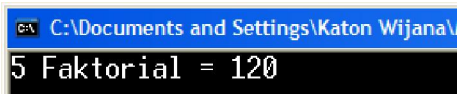
1. Buka Qt Creator dan buat project Qt Console Application baru dengan nama **Labs2**, kemudian tulis kode berikut.

```
#include <QtCore/QCoreApplication>
#include <iostream>

long int kali(long int i); /*prototype*/
int main(int argc, char *argv[])
{
    using namespace std;
    QCoreApplication a(argc, argv);
    int i,n;
    long int fak;
    n = 5;
    /*menghitun n faktorial (5!)*/
    if(n<=0)
        fak=0;
    else
        for(i=1;i<=n;i++)
            fak = kali(i);
    cout << n << " Faktorial = " << fak << endl;
    return a.exec();
}

/*---Fungsi kali---*/
long int kali(long int i)
{
        static long int f=1;
        f = f * i;
}
```

2. Kemudian jalankan kode diatas dengan menekan tombol Ctrl+R, outputnya adalah sebagai berikut.



Analisa Program:

- Dari contoh program ini, variable **f** di fungsi **kali ()** merupakan variable lokal yang bersifat **statik** yang mempunyai nilai awal 1. Pada fungsi ini nilai variabel **f** yang lama akan dikalikan dengan nilai variable i untuk mendapatkan nialai **f** yang baru.

- Pada fungsi utama, fungsi **kali()** akan dipanggil sebanyak **n** kali dengan nilai yang dikirim ke fungsi berupa nilai **1** sampai dengan nilai **n** (pada contoh ini **n** = 5), sehingga akan dihasilkan suatu niali **n!**.
- Supaya nilai variable **f** yang lama masih tetap dipertahankan, maka variable ini perlu dibuat menjadi variable **statik**. Jika variabel ini tidak bersifat static, maka setiap kali fungsi **kali()** dipanggil, nilai variable **f** akan mempunyai nilai awal 1 lagi.

Penggunaan variabel lokal lebih disarankan, karena penggunaan variabel global akan memnyebabkan dampak-dampak sebagai berikut :

- Memboroskan memori computer karena computer masih menyimpan nilainya walaupun sudah tidak diperlukan lagi.
- Mudah terjadi kesalahan program karena satu perubahan dapat menyebabkan perubahan menyeluruh pada program.
- Pembuatan fungsi lebih sulit, karena harus diketahui variable global apa saja yang digunakan.
- Pendeteksian kesalahan program lebih sulit dilakukan.

Pengiriman Parameter

Seperti contoh program-program di atas, fungsi dapat menerima nilai melalui parameter formal dan dapat mengembalikan nilai melalui statment **return**. Ketika fungsi dipanggil, fungsi tersebut akan melakukan suatu pekerjaan dan mengirimkan suatu nilai hasil suatu pekerjaan tersebut yang dinamakan nilai kembalian (*return value*). Jika kita mendeklarasikan seperti berikut:

int fungsiku();

Ini berarti kita mendeklarasikan fungsi bernama **fungsiku** yang akan mengembalikan nilai bertipe integer. Jika kita mendeklarasikan seperti berikut:

int fungsiku(int nilaiInt, float nilaiFloat);

Ini berarti kita mendeklarasikan fungsi bernama **fungsiku** yang juga akan mengembalikan nilai bertipe integer dan selain itu juga menerima 2 buah nilai yang satu bernama **nilaiInt** bertipe **int** dan yang lainnya adalah bernama **nilaiFloat** bertipe **float**. Variabel-variabel penerima nilai ini disebut parameter formal, daftar nilai-nilai yang diterima oleh fungsi ini dinamakan *parameter list*. Pada contoh di atas, *paremeter list* tersebut adalah : **nilaiInt** yaitu sebuah variabel bertipe **int** dan **nilaiFloat** yaitu sebuah variabel bertipe **float**.

Ketika kita mengirimka nilai ke dalam suatu fungsi, yaitu ketika memanggil fungsi sambil menuliskan nilai

yang dikirim di dalam tanda kurung, parameter ini dinamakan parameter aktual atau argumen. Sebagai contoh misalnya :

Hasil = fungsiku(10, 12.5);

Tampak bahwa nilai 10 (bertipe int) dan nilai 12.5 (bertipe float) dikirim sebagai parameter aktual atau argumen, tipe-tipe data dari parameter aktual ini harus sesuai dengan tipe-tipe data yang dideklarasikan pada parameter formal. Pada contoh ini nilai **10** dikirim ke parameter formal pertama dan nilai **12.5** dikirim ke parameter formal kedua dan keduanya sudah sesuai dengan tipe data yang dideklarasikan pada fungsi **fungsiku().**

Pengiriman parameter ke suatu fungsi dapat dilakukan dengan dua cara, yaitu yang disebut pengiriman secara nilai (*by value*) atau pengiriman secara acuan (*by reference*). Pada pengiriman secara nilai, yang dikirimkan adalah nilai (*value*) dari parameter tersebut, jadi pada waktu memanggil fungsi, parameter dapat langsung diisi suatu nilai tidak harus menggunakan suatu variabel, sedangkan pengiriman secara acuan yang dikirimkan adalah alamat dari variabel yang menyimpan nilai yang dikirmkan tersebut.

Hasil dari suatu fungsi dapat diperoleh dari nilai kembaliannya (**return**) atau dengan variabel global. Seperti contoh pada **Lab.4**, hasil proses dari suatu fungsi tersebut dapat diperoleh karena variabel yang dipakai dalam fungsi bersifat global. Selain dengan cara tersebut di atas, hasil dari suatu fungsi dapat juga diperoleh dari parameter aktual yang dikirimkan ke parameter formal, karena parameter formal seolah-olah akan mengirimkan kembali nilai hasil proses dalam fungsi. Pengiriman parameter yang seolah-olah akan mengirimkan kembali nilai hasil proses dalam fungsi ini dinamakan pengiriman parameter secara acuan (pass by reference). Lebih jauh mengenai pengiriman parameter secara acuan ini akan dibahas pada Bab 5 yaitu mengenai Pointer dan References.

Parameter Default

Pada pembahasan sebelumnya, sudah dijelaskan bahwa untuk setiap parameter formal yang telah dideklarasikan pada prototype, harus mendapatkan nilai yang dikirim pada saat pemanggilan fungsi melalui parameter aktual bahkan tipe data dari parameter aktual tersebut harus sesuai dengan tipe data yang dideklarasikan pada parameter formal.

Sebenarnya dengan memberikan nilai default yang dinamakan *default parameter*, suatu parameter formal bisa mempunyai suatu nilai default ketika tidak ada nilai yang diterima dari parameter aktual. Misalnya deklarasi prototype seperti berikut :

int fungsiku(int nilaiInt = 10);

Ini berarti, fungsiku() akan mengembalikan suatu nilai bertipe **int** dan menerima nilai parameter bertipe **int**, jika tidak ada nilai yang diterima maka akan digunakan nilai default yaitu **10**. Karena nama parameter tidak diwajibkan pada prototype, maka prototype tersebut juga boleh ditulis :

int fungsiku(int = 10);

Pemakaian parameter default ini tidak mengubah definisi fungsi, header dari definisi fungsi tersebut tetap seperti berikut:

int fungsiku(int x);

Jika pemanggilan fungsi **fungsiku()** tidak disertai parameter aktual maka kompiler akan memberikan nilai default 10 pada x. Seperti sudah dijelaskan pada Lab.1, nama dari default parameter tidak harus sama dengan nama pada header definisi fungsi, nilai default dikerjakan berdasarkan posisi parameter bukan nama parameter.

Semua parameter fungsi dapat diberikan nilai default, dengan syarat jika tidak ada nilai default untuk parameter di kanannya maka parameter tersebut tidak boleh diberikan nilai default. Misalnya jika prototype suatu fungsi adalah sepoerti berikut:

int fungsiku(int a, int b, int c);

Berarti kita hanya boleh memberikan nilai default untuk b jika kita telah memberikan nilai default untuk c. Nilai default untuk a hanya boleh diberikan jika kita telah memberikan nilai default untuk b dan c.

Labs.6 Default Parameter.

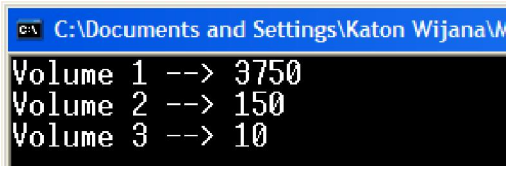
1. Buka Qt Creator dan buat project Qt Console Application baru dengan nama **Labs2**, kemudian tulis kode berikut.

```
#include <QtCore/QCoreApplication>
#include <iostream>
int volume(int,int=1,int=1); /*prototype*/
int main(int argc, char *argv[])
{
    using namespace std;
    QCoreApplication a(argc, argv);
    int panjang,lebar,tinggi;
    panjang = 10;
    lebar = 15;
    tinggi = 25;
    /*menghitun volume*/
    cout << "Volume 1 --> " << volume(panjang,lebar,tinggi)<< endl;
    cout << "Volume 2 --> " << volume(panjang,lebar)<< endl;
    cout << "Volume 3 --> " << volume(panjang)<< endl;
    return a.exec();
}

/*---Fungsi volume---*/
int volume(int p, int l, int t)
{
    return p * l * t;
}
```

2. Kemudian jalankan kode diatas dengan menekan tombol Ctrl+R, outputnya adalah sebagai

berikut.



Analisa Program:

- Dari contoh program ini, Volume 1 dihasilkan dari 10 x 15 x 25 karena semua parameter formal menerima nilai, maka hasilnya **3750**.
- Dari contoh program ini, Volume 1 dihasilkan dari 10 x 15 x 1 karena parameter formal ketiga **tidak** menerima nilai, maka hasilnya 1**50**.
- Dari contoh program ini, Volume 1 dihasilkan dari 10 x 1 x 1 karena parameter formal kedua dan ketiga **tidak** menerima nilai, maka hasilnya 1**0**.