Chapter 5 – Pointer dan References

Agenda

Pada chapter ini kita akan membahas beberapa topik yang berhubungan dengan pointer dan reference yaitu:

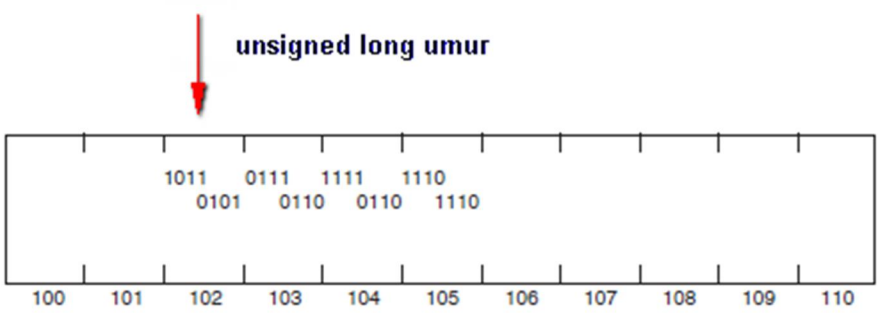
- Penggunaan Pointer.
- Pointer dan Array.
- Mengalokasikan memory dengan keyword 'new'.
- Penggunaan Reference.

Apa itu Pointer?

Pointer adalah variabel yang dapat menyimpan alamat memory. Untuk dapat memahami pointer lebih jauh anda perlu mengenal sedikit tentang memory komputer.

Memory Komputer

Memory Komputer dibagi menjadi beberapa lokasi memory yang berurutan dan mempunyai nomor tertentu. Setiap variabel akan disimpan di lokasi yang unik dalam memory yang disebut alamat memory (*memory address*). Contoh pada gambar dibawah ini menunjukan variabel dengan nama umur yang bertipe *unsigned long*.



Setiap lokasi dalam memory dapat menyimpan data dengan ukuran 1 byte (8 bit), untuk menympan data bertipe *unsigned long* dibutuhkan memory dengan ukuran 4 bytes (32 bit). Dari contoh diatas byte pertama dari variabel umur disimpan pada alamat memory 102, maka alamat memory dari variabel umur adalah 102.

Mengambil Alamat Memory dari Variabel

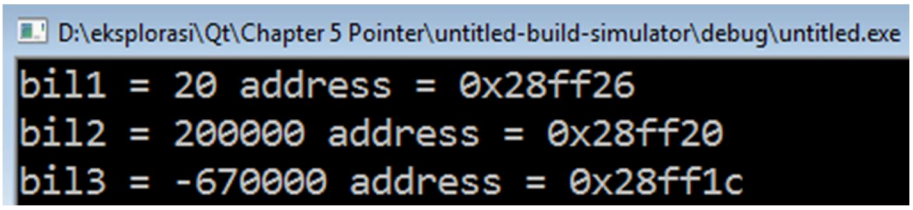
Tiap komputer mempunyai skema yang berbeda untuk penomoran memory, sebagai programmer anda tidak perlu tahu skema alamat dalam memory untuk menyimpan variabel karena kompiler akan melakukan pekerjaan tersebut untuk anda. Jika anda ingin mengetahui pada alamat memory yang mana variabel anda disimpan maka anda dapat menggunakan operator *address-of* (&).

Labs.1 Menampilkan alamat memory menggunakan *address-of* operator.

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

```
#include <QtCore/QCoreApplication>
#include <iostream>
int main(int argc, char *argv[])
{
    using namespace std;
    QCoreApplication a(argc, argv);
    unsigned short bil1 =20;
    ulong bil2 = 200000;
    long bil3 = -670000;
    cout << "bil1 = " << bil1 << " address = " << &bil1 << endl;
    cout << "bil2 = " << bil2 << " address = " << &bil2 << endl;
    cout << "bil3 = " << bil3 << " address = " << &bil3 << endl;
    return a.exec();
}
```

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



Analisa Program :

- Pada program diatas operator address of (&) digunakan untuk mengetahui alamat memory tempat variabel bil disimpan.
- Ketika anda mendeklarasikan variabel dengan tipe tertentu maka compiler akan menentukan ukuran dari memory yang diperlukan untuk menyimpan data dan secara otomatis menetapkan alamat memory dimana variabel tersebut akan disimpan.

Menyimpan Alamat Variabel pada Pointer

Setiap variabel mempunyai alamat, bahkan jika anda tidak tau secara spesifik alamat memory dari variabel tersebut, anda tetap dapat menyimpan alamat variabel kedalam pointer.
Sebagai contoh untuk mendeklarasikan pointer yang menunjuk ke variabel tertentu yang bertipe integer, anda dapat menuliskannya sebagai berikut.

```
int *pBil = 0;
```

Statement diatas bertujuan untuk membuat pointer variabel yang menunjuk ke alamat variabel bertipe integer. Tanda bintang (*) digunakan untuk mendeklarasikan variabel pointer.

Pada contoh diatas pemberian nilai 0 berarti anda mendeklarasikan ***null pointer***, setiap pointer ketika dideklarasikan harus diinisialiasi nilainya. Jika anda belum tahu alamat yang akan ditunjuk oleh pointer maka anda dapat memberi nilai 0. Pointer yang tidak diinisialisasi disebut dengan wild pointer karena bisa menunjuk ke alamat manapun, wild pointer harus dihindari karena sangat berbahaya.

TIPS
Selalu lakukan inisialisasi ketika membuat pointer.

Memberi Nama Pointer

Karena pointer juga merupakan variabel maka aturan penamaan pointer juga sama dengan aturan penamaan variabel biasa. Kesepakatan tidak tertulis programmer dalam pemberian nama pointer adalah diawali dengan huruf p misal (pBil, pUmur).
Contoh dibawah ini adalah cara deklarasi dan inisialiasi pointer.

```
int *pBil = 0; //membuat variabel pointer dan inisialisasi null
int bil = 12; //deklarasi variabel
pBil = &bil; //menunjuk ke alamat variabel bil
```

Pada baris yang ketiga dapat anda lihat bahwa pointer **pBil** menunjuk ke alamat dari variabel **bil**, tanda address-of (&) digunakan untuk mengambil alamat memory dari variabel **bil**. Anda dapat menuliskan statement diatas dengan lebih singkat sebagai berikut:

```
int bil = 12; //deklarasi variable
int *pBil = &bil; //menunjuk ke alamat variabel bil
```

Mengambil Nilai dari Variabel

Mengambil nilai dari variabel dengan menggunakan pointer disebut dengan indirection karena anda secara tidak langsung mengakses nilai dari variabel melalui pointer. Sebagai contoh anda dapat mengakses nilai dari variabel **bil** diatas menggunakan pointer **pBil**.
Operator *indirection* (*) disebut juga dengan operator dereferensi, ketika pointer di dereferensi maka nilai dari variabel yang alamatnya ditunjuk oleh pointer dapat diambil.

```
int number = *pBil; //mengambil nilai variabel yg alamatnya disimpan pada
pointer pBil
```

Pada kode ditas dapat dilihat bahwa nilai dari ***pBil** akan sama dengan nilai **bil**, karena pointer **pBil** mereferensi ke alamat dimana variabel **bil** disimpan, maka number akan bernilai 12.

```
*pBil = 20; //nilai dari variabel bil juga akan berubah menjadi 20
```

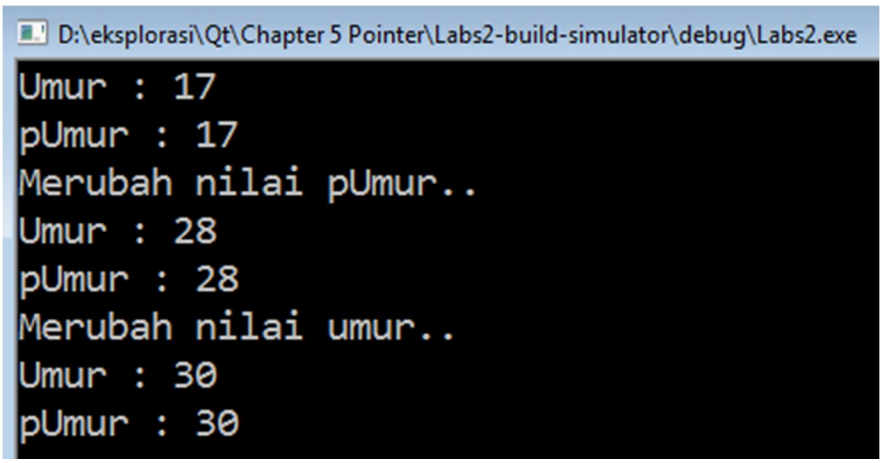
Pada kode diatas nilai dari variabel **bil** akan berubah menjadi 20, karena variabel **bil** direferensi oleh pointer **pBil**.

Labs.2 Memanipulasi data menggunakan Pointer

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

```
#include <QtCore/QCoreApplication>
#include <iostream>
int main(int argc, char *argv[])
{
    using namespace std;
    QCoreApplication a(argc, argv);

    ushort umur;
    ushort *pUmur = 0;
    umur = 17;
    cout << "Umur : " << umur << endl;
    pUmur = &umur;
    cout << "pUmur : " << *pUmur << endl;

    cout << "Merubah nilai pUmur.." << endl;
    *pUmur = 28;
    cout << "Umur : " << umur << endl;
    cout << "pUmur : " << *pUmur << endl;

    cout << "Merubah nilai umur.." << endl;
    umur = 30;
    cout << "Umur : " << umur << endl;
    cout << "pUmur : " << *pUmur << endl;
    return a.exec();
}
```

2. Tekan Ctrl+R untuk menjalankan kode diatas, outputnya adalah sebagai berikut.



Analisa Program:

- Pada program diatas pointer **pUmur** mereferensi/menunjuk ke alamat dimana nilai variabel **umur** disimpan.
- Untuk mengakses nilai dari variabel **umur** lewat pointer dapat menggunakan dereference operator (*).
- Ketika nilai dereference pointer ***pUmur** diubah menjadi 28, maka akan mempengaruhi nilai pada variabel **umur** yang akan menjadi 28 juga.
- Ketika nilai variabel **umur** diubah menjadi 30, dan anda mengakses nilainya dengan menggunakan pointer ***pUmur** maka nilainya juga akan berubah menjadi 30.

Mengganti alamat yang direferensi oleh Pointer

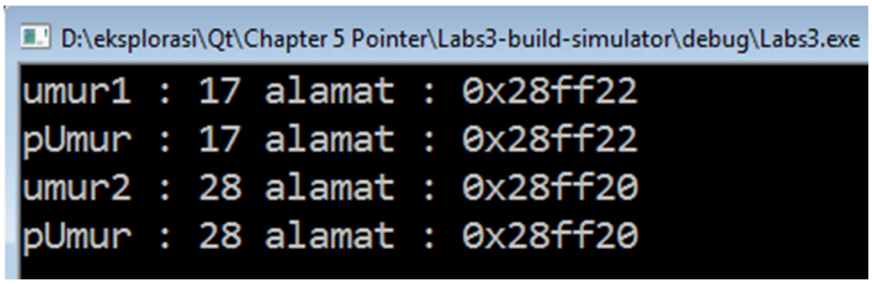
Anda juga dapat mengganti alamat variabel yang direferensi oleh pointer tertentu tanpa harus mengetahui nilai dari variabel tersebut.

Labs.3 Mengganti alamat yang di referensi oleh pointer

1. Buat project Qt Console Application baru, beri nama Labs3, kemudian tulis kode berikut

```
#include <QtCore/QCoreApplication>
#include <iostream>
int main(int argc, char *argv[])
{
    using namespace std;
    QCoreApplication a(argc, argv);

    ushort umur1 = 17, umur2 = 28;

    ushort *pUmur = &umur1; //ganti referensi
    cout << "umur1 : " << umur1 << " alamat : " << &umur1 << endl;
    cout << "pUmur : " << *pUmur << " alamat : " << pUmur << endl;

    pUmur = &umur2;
    cout << "umur2 : " << umur2 << " alamat : " << &umur2 << endl;
    cout << "pUmur : " << *pUmur << " alamat : " << pUmur << endl;
    return a.exec();
}
```

2. Tekan Ctrl+R untuk menjalankan program diatas, outputnya adalah sebagai berikut.



Analisa:

- Pada program diatas dapat dilihat bahwa pertama kali pointer **pUmur** mereferensi pada alamat variabel **umur1**, sehingga ketika dicetak nilai dari ***pUmur** sama dengan nilai variabel **umur1**.
- Anda dapat mengganti referensi dari **pUmur** yang tadinya menunjuk ke alamat variabel **umur1** menjadi menunjuk ke alamat variabel **umur2**, sehingga ketika ***pUmur** dicetak menghasilkan nilai yang sama dengan variabel **umur2**.

Pointer dan Array

Pada C++ nama dari array adalah konstan pointer yang menunjuk pada elemen pertama dari array, misal untuk deklarasi array berikut

```
int Numbers[5];
```

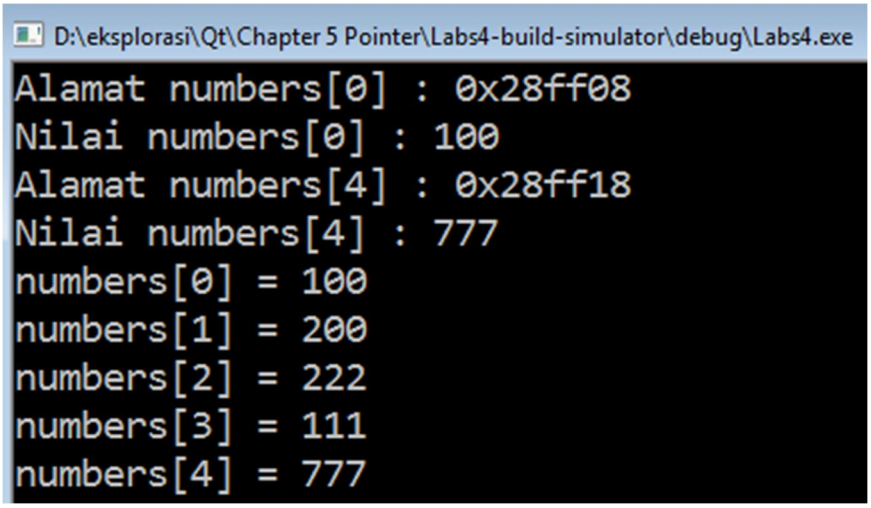
Numbers adalah pointer yang menunjuk alamat **&Numbers[0]** yang merupakan alamat dari elemen pertama array diatas.
Anda dapat menggunakan nama array sebagai konstan pointer, misalnya **Numbers+3** adalah cara penulisan untuk mengakses pointer yang menunjuk ke **Numbers[3]**.

Labs.4 Pointer dan Array

1. Buat project Qt Console Application dengan nama **Labs4**, kemudian tulis kode berikut.

```
#include <QtCore/QCoreApplication>
#include <iostream>
int main(int argc, char *argv[])
{
    using namespace std;
    QCoreApplication a(argc, argv);

    const int ARRAY_LENGTH = 5;
    int numbers[ARRAY_LENGTH] = {100,200,222,111,777};

    //mengakses alamat pertama dari array (numbers[0])
    cout << "Alamat numbers[0] : " << numbers << endl;
    //mengakses nilai dari elemen pertama array (numbers[0])
    cout << "Nilai numbers[0] : " << *numbers << endl;

    //mengakses alamat numbers[4]
    cout << "Alamat numbers[4] : " << numbers+4 << endl;
    //mengakses nilai dari numbers[4]
    cout << "Nilai numbers[4] : " << *(numbers+4) << endl;

    const int *pNumber = numbers;
    //menggunakan pointer untuk mencetak semua elemen array
    for(int i=0; i<ARRAY_LENGTH; i++)
    {
        cout << "numbers[" << i << "] = " << *(pNumber+i) << endl;
    }
    return a.exec();
}
```

2. Tekan Ctrl+R untuk menjalankan program, outputnya adalah sebagai berikut.



Analisa :
Nama dari array **numbers** merupakan konstan pointer yang menunjuk alamat element pertama pada array (**numbers[0]**), jadi jika anda ingin mengetahui nilai dari elemen pertama array anda dapat menggunakan *dereference* operator ***numbers**.
Anda dapat menggunakan nama array **numbers+4** untuk menunjuk ke alamat elemen **numbers[4]**, untuk menampilkan nilai **numbers[4]** anda dapat menuliskan ***(numbers+4)**.

Kapan kita menggunakan pointer?

Setelah kita mempelajari cara penggunaan pointer sekarang kita akan melihat kapan pointer biasa digunakan dalam pemrograman.

- Pengaturan data pada *free store* / *heap memory*.
- Mengakses class member dan data function.
- Passing variabel dengan reference pada function.

Mengalokasikan tempat dengan keyword 'new'

Anda dapat mengalokasikan memory pada *free store* / *heap memory* dengan menggunakan keyword 'new' diikuti dengan tipe data dari objek yang akan anda simpan sehingga compiler dapat mengetahui berapa banyak memory yang dibutuhkan untuk menyimpan data tersebut. Contoh penggunaan keyword 'new' dapat dilihat pada kode berikut:

```
//mengalokasikan memory di heap untuk menyimpan data integer
int *pBil = new int;

//nilai 19 akan disimpan di heap yg sudah dialokasikan
*pBil = 19;
```

Membersihkan memory dengan keyword 'delete'

Ketika anda sudah selesai menggunakan objek yang ada di memory, anda harus mengosongkan kembali memory tersebut agar dapat digunakan kembali. Anda dapat menggunakan keyword 'delete' untuk mengembalikan memory yang anda gunakan ke *heap* / *free store*.

Penting untuk anda ketahui bahwa memory yang dialokasikan menggunakan keyword 'new' tidak akan dibersihkan secara otomatis, maka sebagai programmer anda harus disiplin untuk membebaskan memory yang sudah tidak digunakan.

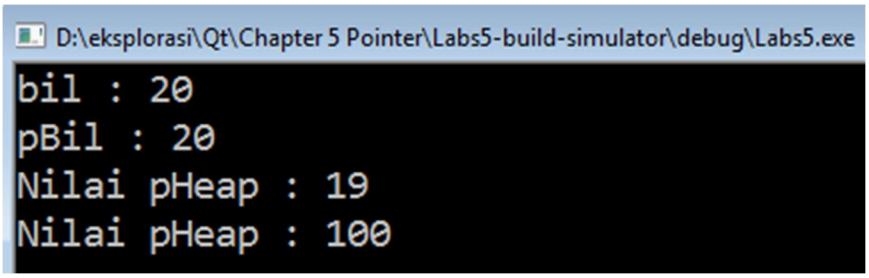
Ketika anda menghapus memory maka pointer tetap menunjuk ke alamat memory yang sudah anda hapus, agar tidak terjadi kesalahan setelah menghapus memory anda disarankan untuk memberi nilai **`null`** (**`0`**) pada pointer.

Labs.5 Mengalokasikan, menggunakan, dan mendelete Pointer

1. Buat project Qt Console Application dengan nama **Labs5**, kemudian tulis kode berikut:

```
#include <QtCore/QCoreApplication>
#include <iostream>
int main(int argc, char *argv[])
{
    using namespace std;
    QCoreApplication a(argc, argv);
    int bil = 20;
    //pointer yang menunjuk ke alamat lokal
    int *pBil = &bil;
    cout << "bil : " << bil << endl;
    cout << "pBil : " << *pBil << endl;
    //mengalokasikan memory di heap untuk menyimpan data integer
    int *pHeap = new int;
    //nilai 19 akan disimpan di heap yg sudah dialokasikan
    *pHeap = 19;
    cout << "Nilai pHeap : " << *pHeap << endl;
    delete pHeap;
    pHeap = 0; //null pointer

    //mengalokasikan memory
    pHeap = new int;
    *pHeap = 100;
    cout << "Nilai pHeap : " << *pHeap << endl;
    delete pHeap;

    return a.exec();
}
```

2. Tekan Ctrl+R untuk menjalankan program, outputnya adalah sebagai berikut.



Analisa:

- **pHeap** adalah pointer yang menunjuk ke alamat memory yang sudah dialokasikan dengan keyword '**new**', anda dapat menyimpan nilai kedalam memory yang dialokasikan dengan***pHeap=19**
- Setelah selesai digunakan anda harus membersihkan memory dengan menggunakan keyword '**delete**', jangan lupa menginisialisasi pointer dengan **null (0)** agar tidak terus menunjuk ke alamat memory yang sudah dihapus.

TIPS

*Setelah menghapus objek di memory dengan keyword **delete** anda harus menginisialisasi pointer yang sudah tidak digunakan dengan nilai **null** (0).*

Membuat objek pada heap

Selain tipe data primitive (int, float, byte, dll) anda juga dapat menyimpan data bertipe class kedalam free store / heap, misal jika anda ingin membuat objek bertipe class Mahasiswa anda dapat mendeklarasikan pointer untuk class tersebut dan mengalokasikan memory di heap untuk menyimpan objek tersebut. Sintaks penulisanya sama dengan sebelumnya.

```
Mahasiswa *mhs = new Mahasiswa;
```

Ketika anda menggunakan keyword 'new' untuk membuat pointer yang menunjuk ke objek maka otomatis default konstruktor dari class tersebut akan dipanggil.

Ketika anda menghapus pointer yang menunjuk ke objek dengan keyword 'delete', maka destruktor akan dipanggil, ini akan memberi kesempatan bagi programmer untuk membersihkan heap memory dari variabel yang sudah tidak digunakan.

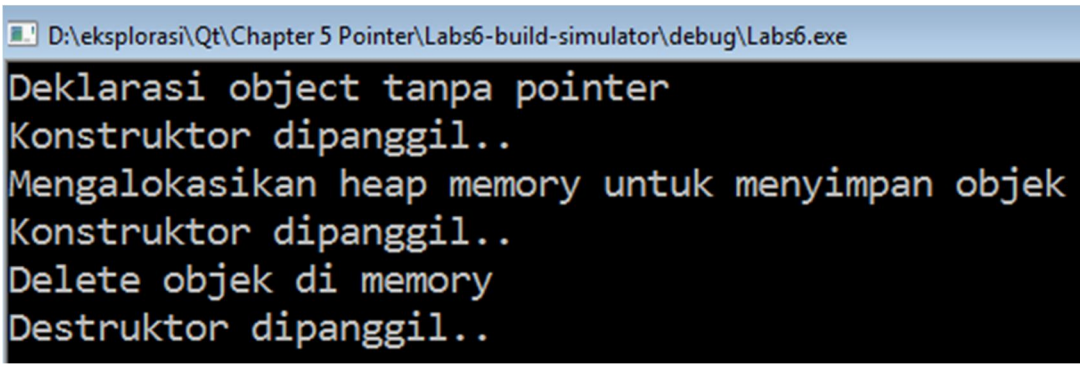
Labs.6 Membuat dan menghapus objek dari Heap

1. Buat project Qt Console Application dengan nama **Labs6**, kemudian tulis kode berikut:

```
#include <QtCore/QCoreApplication>
#include <iostream>
using namespace std;
class Mahasiswa
{
```

```
public:
    Mahasiswa();
    ~Mahasiswa();
private:
    float ipk;
};
Mahasiswa::Mahasiswa()
{
    cout << "Konstruktor dipanggil.." << endl;
    ipk=3.5;
}
Mahasiswa::~Mahasiswa()
{
    cout << "Destruktor dipanggil.." << endl;
}
int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    cout << "Deklarasi object tanpa pointer " << endl;
    Mahasiswa mhs1;
    cout << "Mengalokasikan heap memory untuk menyimpan objek " << endl;
    Mahasiswa *mhs2 = new Mahasiswa;
    cout << "Delete objek di memory " << endl;
    delete mhs2;
    mhs2 = 0; //null pointer
    return a.exec();
}
```

2. Tekan Ctrl+R untuk menjalankan program, outputnya adalah sebagai berikut.



Analisa :

- Pada program diatas kita membuat class Mahasiswa yang mempunyai objek konstruktor dan destruktor.
- Pertama kali kita mendeklarasikan object **`mhs1`** pada local variable (stack), pembuatan object ini menyebabkan konstruktor dipanggil.

- Kemudian dibuat pointer yang menunjuk ke objek di heap dengan nama **mhs2**, ketika objek **mhs2** dibuat, objek konstruktor dipanggil. Ketika anda menghapus objek di heap menggunakan **delete** maka objek destruktor akan dipanggil.
- Objek desktruktor untuk **mhs1** akan dipanggil ketika fungsi main berakhir.

Menggunakan const Pointer

Anda dapat menggunakan keyword 'const' pada pointer dengan menuliskannya sebelum atau sesudah tipe data, atau keduanya. Contoh deklarasi const pointer dapat dilihat pada kode dibawah ini:

```
const int * pBil1;
int * const pBil2;
const int * const pBil3;
```

Tiga statement diatas memiliki pengertian yang berbeda, yaitu:

- Statement pertama : **pBil1** adalah pointer yang menunjuk ke konstan integer, jadi nilai yang ditunjuk oleh pointer tidak dapat diubah.
- Statement kedua : **pBil2** adalah konstan pointer yang menunjuk ke variabel integer, nilai variabel integer dapat diubah namun **pBil2** tidak dapat menunjuk ke variabel lain.
- Statement ketiga : pBil3 adalah konstan pointer yang menunjuk ke konstan variabel bertipe integer, nilai variabel tidak dapat diubah dan pointer pBil3 tidak dapat menunjuk ke variabel lain.

TIPS
Lihat letak penulisan keyword const, jika sebelum tipe data maka nilai konstan, jika setelah tipe data maka alamat pointer yang konstan.

Apa itu Reference

Pada pembahasan sebelumnya kita membahas penggunaan pointer untuk mengakses objek secara tidak langsung (indirect). Fungsi reference mirip seperti pointer namun dengan penulisan yang relatif lebih mudah.
Reference adalah alias, ketika anda membuat reference anda menginisialisasi dengan nama dari objek yg dijadikan target. Reference adalah alternatif nama dari objek target, jika anda merubah reference maka objek target juga akan berubah.
Cara penulisan reference adalah menambahkan operator (&) didepan nama variabel, contohnya :

```
int &rBil = intBil;
```

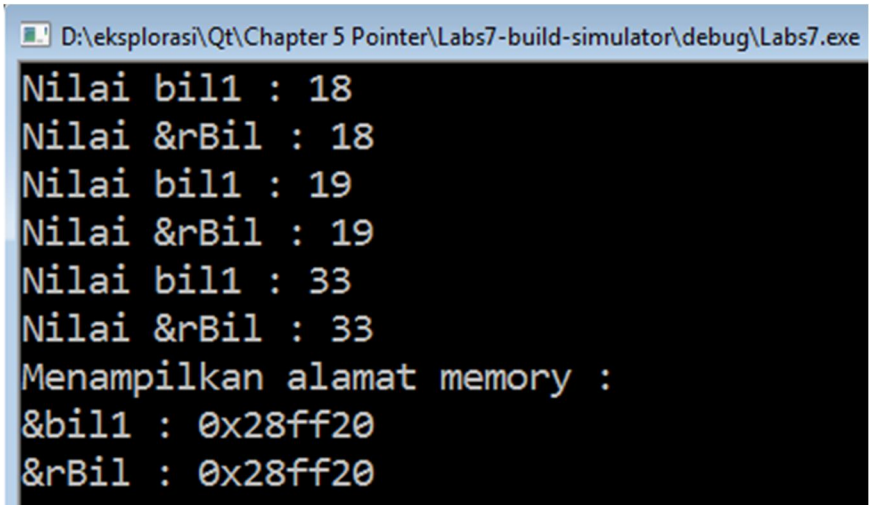
Statement diatas dapat diartikan "rBil adalah referensi dari variabel intBil", reference berbeda dengan variabel biasa karena reference harus diinisialisasi ketika dibuat.

Labs.7 Membuat dan Menggunakan Reference

1. Buat project Qt Console Application dengan nama **Labs7**, kemudian tulis kode berikut:

```
#include <QtCore/QCoreApplication>
#include <iostream>
using namespace std;
int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    int bil1 = 18;
    int &rBil = bil1;
    cout << "Nilai bil1 : " << bil1 << endl;
    cout << "Nilai &rBil : " << rBil << endl;

    bil1 = 19;
    cout << "Nilai bil1 : " << bil1 << endl;
    cout << "Nilai &rBil : " << rBil << endl;

    rBil = 33;
    cout << "Nilai bil1 : " << bil1 << endl;
    cout << "Nilai &rBil : " << rBil << endl;

    cout << "Menampilkan alamat memory :" << endl;
    cout << "&bil1 : " << &bil1 << endl;
    cout << "&rBil : " << &rBil << endl;
    return a.exec();
}
```

2. T ekan Ctrl+R untuk menjalankan program, outputnya adalah sebagai berikut.



Analisa:

- Pertama kita mendeklarasikan referensi **`rBil=bil1`**, maka ketika dicetak nilai **`rBil`** sama dengan nilai variabel **`bil1`** karena **`rBil`** merupakan reference / alias dari **`bil1`**.
- Ketika variabel **`bil1`** nilainya dirubah menjadi 19, maka otomatis nilai dari **`rBil`** juga berubah menjadi 19.

- Demikian pula ketika **rBil** nilainya dirubah menjadi 33, maka nilai dari **bil1** juga ikut berubah.
- Anda juga dapat menampilkan alamat memory dari variabel dan variabel reference dengan menambahkan keyword (&) didepan variabel.

Re-assign Reference Variable

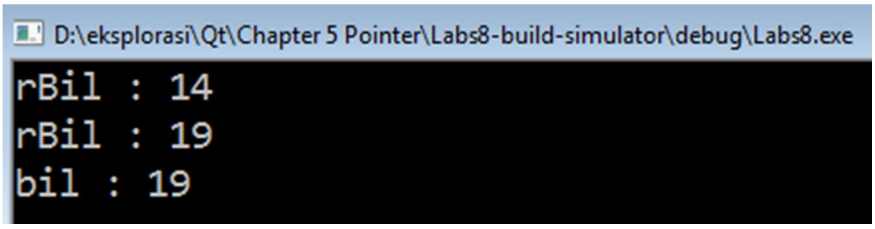
Variabel reference tidak dapat di re-assign (ditetapkan ulang). Agar lebih jelas perhatikan contoh dibawah ini:

Labs.8 Re-assign Reference Value

1. Buat project Qt Console Application dengan nama **Labs8**, kemudian tulis kode berikut:

```
#include <QtCore/QCoreApplication>
#include <iostream>
int main(int argc, char *argv[])
{
    using namespace std;
    QCoreApplication a(argc, argv);
    int bil = 14;
    int &rBil = bil;
    cout << "rBil : " << rBil << endl;

    int bil2 = 19;
    rBil = bil2; //tebak hasilnya !
    cout << "rBil : " << rBil << endl;
    cout << "bil : " << bil << endl;
    return a.exec();
}
```

2. T ekan Ctrl+R untuk menjalankan program, outputnya adalah sebagai berikut.



Analisa:
Variabel reference **rBil** yang sudah diinisialisasi dengan **bil1** coba di re-assign dengan **bil2** dan gagal, karena **rBil=bil2** tidak menjadikan referensinya berubah tetapi nilai **bil2** yang mengganti nilai **rBil** dan **bil1**.

Passing function argument dengan reference

Pada chapter sebelumnya tentang function, kita sudah membahas beberapa keterbatasan dari function diantaranya, argument hanya dapat di-passing by value, dan return statement hanya dapat mengembalikan satu nilai saja.

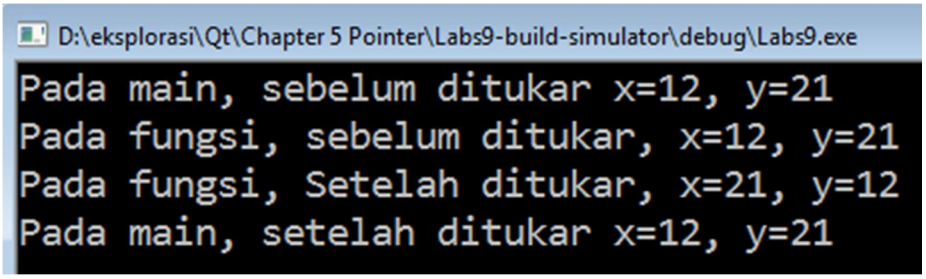
Passing reference value pada function dapat mengatasi masalah diatas. Contoh dibawah ini akan menunjukan perbedaan penggunaan passing by value dan passing by reference (dengan pointer dan variaabel reference).

Labs.9 Passing by Value

1. Buat project Qt Console Application dengan nama **Labs9**, kemudian tulis kode berikut:

```
#include <QtCore/QCoreApplication>
#include <iostream>
using namespace std;
void Tukar(int x,int y);
int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    int x=12, y=21;
    cout << "Pada main, sebelum ditukar x=" << x << ", y=" << y << endl;
    Tukar(x,y);
    cout << "Pada main, setelah ditukar x=" << x << ", y=" << y << endl;
    return a.exec();
}
void Tukar(int x,int y)
{
    int tampung;
    cout << "Pada fungsi, sebelum ditukar, x=" << x << ", y=" << y << endl;
    tampung = x;
    x=y;
    y=tampung;
    cout << "Pada fungsi, Setelah ditukar, x=" << x << ", y=" << y << endl;
}
```

2. T ekan Ctrl+R untuk menjalankan program, outputnya adalah sebagai berikut.



Analisa

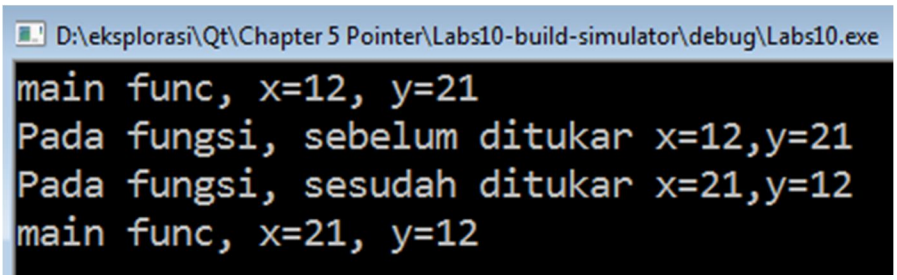
- Pada kode diatas dapat dilihat bahwa passing by value ke fungsi **Tukar()** tidak akan mempengaruhi variabel **x** dan **y** yang ada pada fungsi **main**, dan hanya berpengaruh pada scope fungsi **Tukar()** saja.

Labs.10 Passing by reference dengan pointer

1. Buat project Qt Console Application dengan nama **Labs10**, kemudian tulis kode berikut:

```
#include <QtCore/QCoreApplication>
#include <iostream>
using namespace std;
void Tukar(int *x, int *y);
int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    int x=12, y=21;
    cout << "main func, x=" << x << ", y=" << y << endl;
    Tukar(&x,&y);
    cout << "main func, x=" << x << ", y=" << y << endl;
    return a.exec();
}
void Tukar(int *x, int *y)
{
    int tampung;
    cout << "Pada fungsi, sebelum ditukar x=" << *x << ",y=" << *y << endl;
    tampung = *x;
    *x = *y;
    *y = tampung;
    cout << "Pada fungsi, sesudah ditukar x=" << *x << ",y=" << *y << endl;
}
```

2. T ekan Ctrl+R untuk menjalankan program, outputnya adalah sebagai berikut.



Analisa:

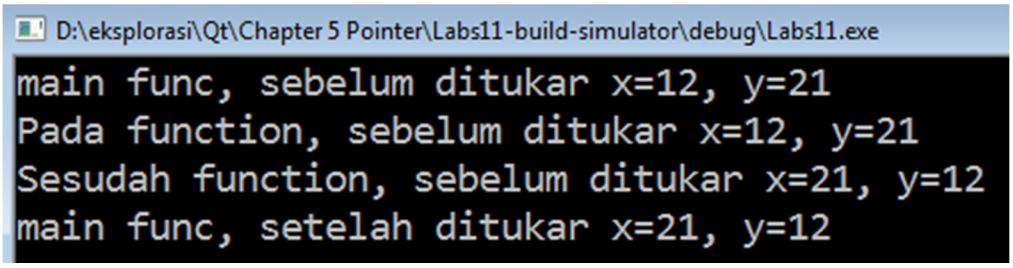
- Pada kode diatas kita melakukan passing by reference untuk passing parameter ke fungsi **Tukar()** menggunakan pointer, dapat anda lihat bahwa setelah fungsi **Tukar()** dijalankan variabel **x** dan **y** di main function nilainya sudah berhasil ditukar.

Labs.11 Menjalankan fungsi Tukar() dengan reference

1. Buat project Qt Console Application dengan nama **Labs11**, kemudian tulis kode berikut:

```
#include <QtCore/QCoreApplication>
#include <iostream>
using namespace std;

void Tukar(int &x, int &y);

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    int x=12, y=21;
    cout << "main func, sebelum ditukar x=" << x << ", y=" << y << endl;
    Tukar(x,y);
    cout << "main func, setelah ditukar x=" << x << ", y=" << y << endl;
    return a.exec();
}

void Tukar(int &x, int &y)
{
    int tampung;
    cout << "Pada function, sebelum ditukar x=" << x << ", y=" << y << endl;
    tampung = x;
    x = y;
    y = tampung;
    cout << "Sesudah function, sebelum ditukar x=" << x << ", y=" << y << endl;
}
```

2. T ekan Ctrl+R untuk menjalankan program, outputnya adalah sebagai berikut.



Analisa:
Pada kode diatas kita juga berhasil menukar nilai **x** dan **y** menggunakan fungsi tukar sama dengan kode sebelumnya. Ini karena passing parameter menggunakan variabel reference.

Function yang mengembalikan beberapa nilai

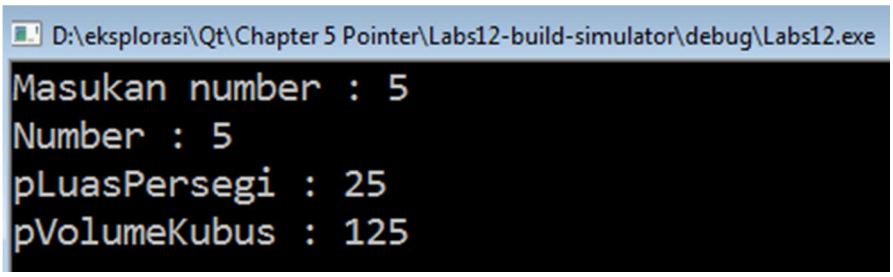
Seperti yang sudah kita bahas sebelumnya bahwa salah satu keterbatasan dari function adalah hanya dapat mengembalikan satu nilai saja. Bagaimana jika anda ingin mengembalikan lebih dari satu nilai pada function? Untuk memecahkan masalah tersebut anda dapat menggunakan function pass by reference. Karena function pass by reference dapat memanipulasi objek asli. Agar lebih jelas coba kerjakan contoh dibawah ini.

Labs.12 Mengembalikan beberapa nilai dengan pointer

1. Buat project Qt Console Application dengan nama **Labs12**, kemudian tulis kode berikut:

```
#include <QtCore/QCoreApplication>
#include <iostream>
using namespace std;
int Hitung(int number, int *pLuasPersegi, int *pVolumeKubus);
int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    int number, pLuasPersegi, pVolumeKubus;
    short error;
    cout << "Masukan number : ";
    cin >> number;
    error = Hitung(number,&pLuasPersegi,&pVolumeKubus);
    if(!error)
    {
        cout << "Number : " << number << endl;
        cout << "pLuasPersegi : " << pLuasPersegi << endl;
        cout << "pVolumeKubus : " << pVolumeKubus << endl;
    }
    else
        cout << "Terjadi Error !! ";
    return a.exec();
}
int Hitung(int number, int *pLuasPersegi, int *pVolumeKubus)
{
    short status;
    if(number > 0)
    {
        *pLuasPersegi = number * number;
        *pVolumeKubus = number * number * number;
        status = 0;
    }
    else
    {
        status = 1;
    }
    return status;
}
```

2. T ekan Ctrl+R untuk menjalankan program, outputnya adalah sebagai berikut.



Analisa:

- Inputan untuk variabel number harus lebih besar dari 0, jika tidak program akan menghasilkan pesan error.
- Dapat dilihat bahwa function **`Hitung()`** mengembalikan 3 nilai yaitu : nilai kembalian dari function itu sendiri yang bertipe integer, **`pLuasPersegi`**, dan **`pVolumeKubus`** yang merupakan parameter bertipe pointer.
- **`pLuasPersegi`** dan **`pVolumeKubus`** nilainya dapat bukan karena nilai kembalian dari function, tapi karena parameter by reference dari function yang berupa pointer, sehingga ketika nilai **`pLuasPersegi`** dan **`pVolumeKubus`** diubah di dalam function nilai variabel asli di main function juga berubah.

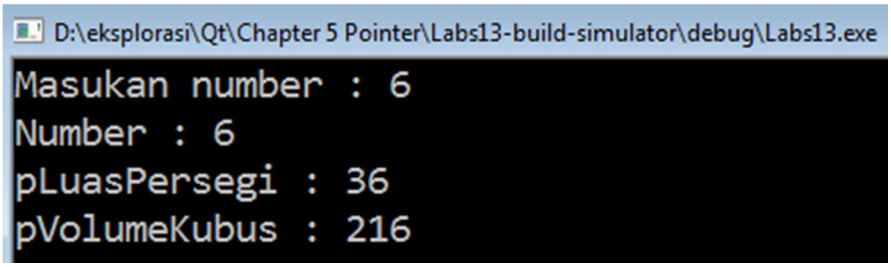
Labs.13 Mengembalikan beberapa nilai dengan reference variabel

1. Buat project Qt Console Application dengan nama **Labs13**, kemudian tulis kode berikut:

```
#include <QtCore/QCoreApplication>
#include <iostream>
using namespace std;

enum ERR_STATUS {SUCCESS, ERROR};
ERR_STATUS Hitung(int,int &,int &);

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    ERR_STATUS status;
    int number, rLuasPersegi, rVolumeKubus;

    cout << "Masukan number : ";
    cin >> number;
    status = Hitung(number,rLuasPersegi,rVolumeKubus);

    if(status==SUCCESS)
    {
        cout << "Number : " << number << endl;
        cout << "pLuasPersegi : " << rLuasPersegi << endl;
        cout << "pVolumeKubus : " << rVolumeKubus << endl;
    }
    else
        cout << "Terjadi Error !!";
    return a.exec();
}

ERR_STATUS Hitung(int number, int &rLuasPersegi, int &rVolumeKubus)
{
    ERR_STATUS status;
    if(number > 0)
    {
        rLuasPersegi = number * number;
        rVolumeKubus = number * number * number;
        status = SUCCESS;
```

```
    }
    else
        status = ERROR;
    return status;
}
```

2. T ekan Ctrl+R untuk menjalankan program, outputnya adalah sebagai berikut.



Analisa:

- Hasil program diatas sama dengan **Labs12** sebelumnya, namun perbedaannya adalah program diatas menggunakan parameter reference pada function **`Hitung()`** sehingga ketika variabel **`rLuasPersegi`** dan **`rVolumeKubus`** pada function diubah nilainya maka variabel di function main juga ikut berubah.
- Keyword **`enum`** digunakan untuk membuat objek enumerasi untuk mempermudah pembacaan program.

Passing By Reference untuk Efisiensi

Setiap kali anda melakukan passing objek by value, copy dari objek tersebut akan dibuat kembali. Untuk tipe data objek yang besar (struct atau class yang dibuat sendri oleh user) ini akan menurunkan performa dari program. Untuk melakukan passing parameter objek melalui function disarankan menggunakan reference pada objek.

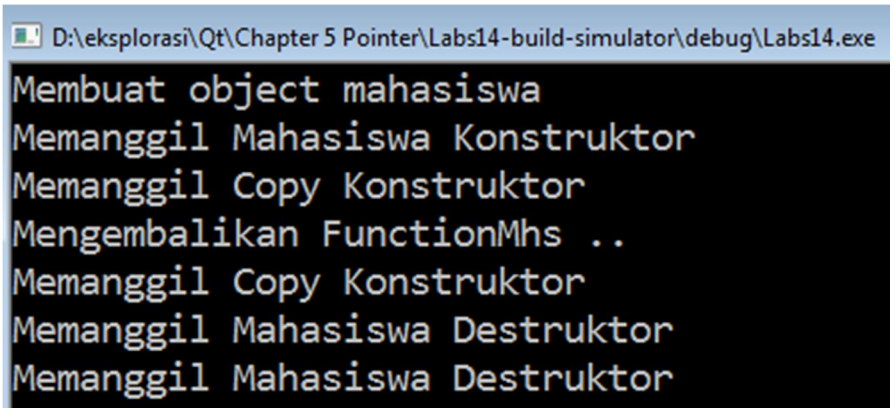
Labs.14 Passing Object By Value

1. Buat project Qt Console Application dengan nama **Labs14**, kemudian tulis kode berikut:

```
#include <QtCore/QCoreApplication>
#include <iostream>
using namespace std;
class Mahasiswa
{
public:
    Mahasiswa();
    Mahasiswa(Mahasiswa&);
    ~Mahasiswa();
};
Mahasiswa::Mahasiswa()
{
    cout << "Memanggil Mahasiswa Konstruktor " << endl;
```

```
}
Mahasiswa::Mahasiswa(Mahasiswa &)
{
    cout << "Memanggil Copy Konstruktor " << endl;
}
Mahasiswa::~Mahasiswa()
{
    cout << "Memanggil Mahasiswa Destruktor " << endl;
}
Mahasiswa FunctionMhs(Mahasiswa objMhs);
int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    cout << "Membuat object mahasiswa " << endl;
    Mahasiswa objMhs1;
    FunctionMhs(objMhs1);
    return a.exec();
}
Mahasiswa FunctionMhs(Mahasiswa objMhs)
{
    cout << "Mengembalikan FunctionMhs .." << endl;
    return objMhs;
}
```

2. T ekan Ctrl+R untuk menjalankan program, outputnya adalah sebagai berikut.



Analisa:

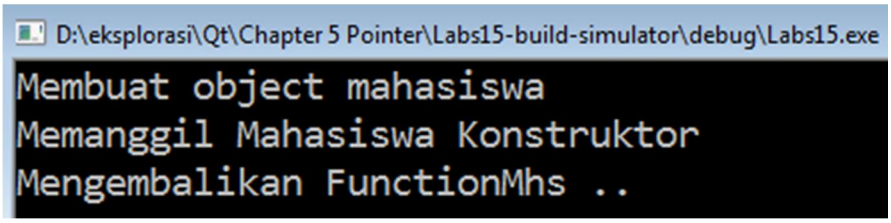
- Dapat kita lihat diatas bahwa passing object by value tidak efisien karena setiap kali function dipanggil dan mengembalikan nilai harus melakukan copy terhadap objek **`objMhs1`**.
- Hal ini dapat dilihat dari output yang dihasilkan, copy konstruktor dipanggil sebanyak 2 kali, saat pemanggilan function dan pengembalian nilai function.
- Cara yang lebih efisien akan dibahas pada contoh program selanjutnya.

Labs.15 Passing Object By Reference

1. Buat project Qt Console Application dengan nama **Labs15**, kemudian tulis kode berikut:

```
#include <QtCore/QCoreApplication>
#include <iostream>
using namespace std;
class Mahasiswa
{
public:
    Mahasiswa();
    Mahasiswa(Mahasiswa&);
    ~Mahasiswa();
};
Mahasiswa::Mahasiswa()
{
    cout << "Memanggil Mahasiswa Konstruktor " << endl;
}
Mahasiswa::Mahasiswa(Mahasiswa &)
{
    cout << "Memanggil Copy Konstruktor " << endl;
}
Mahasiswa::~Mahasiswa()
{
    cout << "Memanggil Mahasiswa Destruktor " << endl;
}
Mahasiswa &FunctionMhs(Mahasiswa &objMhs);
int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    cout << "Membuat object mahasiswa " << endl;
    Mahasiswa objMhs1;
    FunctionMhs(objMhs1);
    return a.exec();
}
Mahasiswa &FunctionMhs(Mahasiswa &objMhs)
{
    cout << "Mengembalikan FunctionMhs .." << endl;
    return objMhs;
}
```

2. T ekan Ctrl+R untuk menjalankan program, outputnya adalah sebagai berikut.



Analisa:

- Dengan menambahkan reference pada function dan parameter yang dikirimkan, performa aplikasi anda dapat lebih efektif karena objek tidak perlu dicopy ketika function dijalankan dan saat function ter sebut mengembalikan nilai.
- Output yang dihasilkan lebih sedikit karena tidak perlu memanggil copy objek konstruktor.