BAB X. Struct

Dalam C++, kita dapat membuat sebuah "tipe data" baru. Maka penulisan variabel baru kita akan menjadi:

```
tipe_data_baru nama_variabel;
```

Misalnya seperti di bawah ini:

```
tipe_baru_guweh variabel_guweh;
```

Hal ini, salah satunya, dapat dilakukan dengan **`struct`**. Syntak dari struct adalah sebagai berikut:

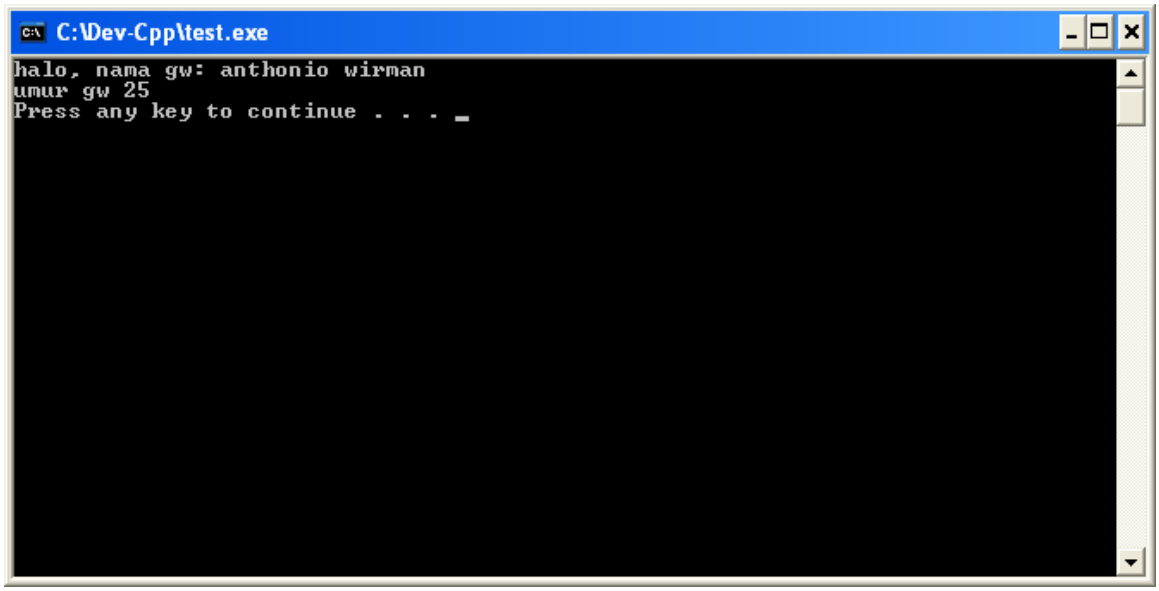
```
struct nama_struct {
//member, dan lain – lain diletakkan disini
};
```

Perhatikan contoh dibawah:

```
#include <iostream>
#include <string>
using namespace std;

struct tipe_baru_guweh {
    string nama_depan;
    string nama_belakang;
    int umur;
}; //semicolon (titik koma) setelah definisi struct
```

```
void main () {
    tipe_baru_guweh variabel_guweh;
    variabel_guweh.nama_depan = "anthonio";
    variabel_guweh.nama_belakang = "wirman";
    variabel_guweh.umur = 25;
    cout << "halo, nama gw: " << variabel_guweh.nama_depan << " "
         << variabel_guweh.nama_belakang << "\n";
    cout << "umur gw " << variabel_guweh.umur << "\n";
}
```

Hasilnya:



Member dari tipe data baru yang dibuat dari struct hanya bisa diakses oleh *object* dari tipe data tersebut. Misalnya, `variabel_guweh` yang merupakan object dari `tipe_baru_guweh`. Object dari struct mengakses membernya dengan menggunakan

tanda titik (“.”), misalnya `umur` yang merupakan member dari `tipe_baru_guweh`, diakses dengan cara:

```
variabel_baru_guweh.umur
```

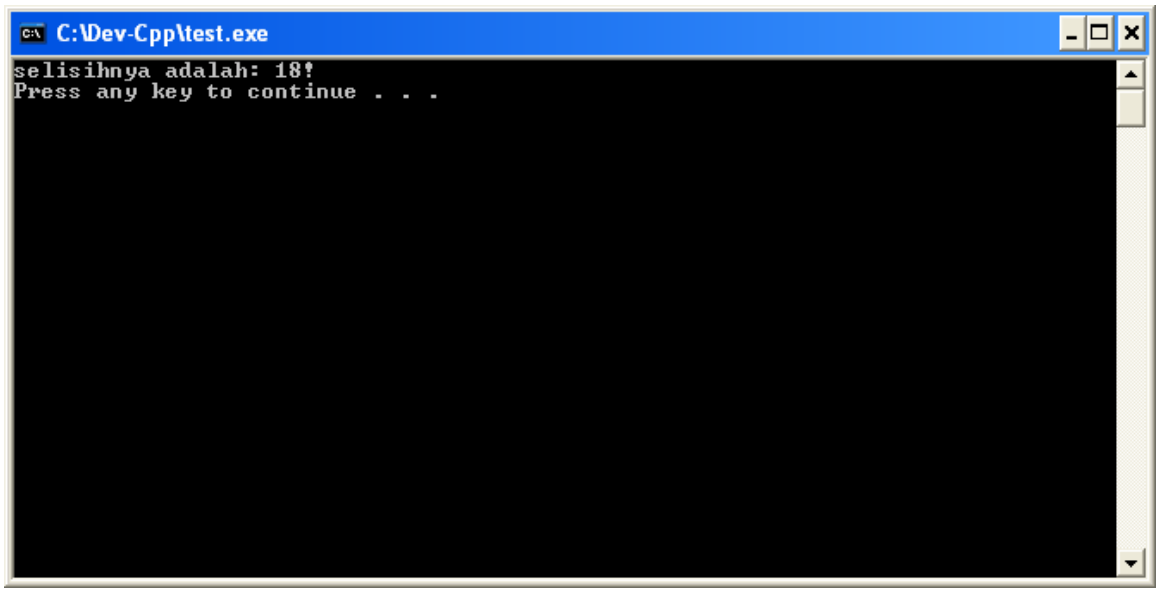
Begitu juga dengan member – member lainnya.

Kita juga bisa menaruh fungsi di dalam struct. Misalnya, kita mau menghitung selisih umur antara dua buah variabel dari tipe data yang sama:

```
#include <iostream>
using namespace std;

struct manusia {
    int umur;
    int selisih_umur (manusia n, manusia m) {
        if (n.umur < m.umur)
                return (m.umur - n.umur);
        else
                return (n.umur - m.umur);
    }
};

void main () {
    manusia manusia1, manusia2;
    manusia1.umur = 34;
    manusia2.umur = 16;
   cout << "selisihnya adalah: " << manusia1.selisih_umur
          (manusia1, manusia2) << "!\n";
 }
```

Disini, `selisih_umur` menerima dua argumen dengan tipe `manusia`. Kemudian membandingkan dua member dari argumen tersebut (`umur`) dan mengembalikan hasil

pengurangannya / selisihnya, berdasarkan salah satu dari 2 kondisi yang dipersiapkan, yang bertipe integer.



Kode di atas dapat ditulis dengan cara lain dengan hasil yang sama, yaitu:

```
#include <iostream>
using namespace std;

struct manusia {
    int umur;
    int selisih_umur (manusia n, manusia m); //prototype fungsi
};

int manusia::selisih_umur (manusia n, manusia m) {
        if (n.umur < m.umur)
                return (m.umur - n.umur);
```

```
        else
                return (n.umur - m.umur);
}

void main () {
    manusia manusia1, manusia2;
    manusia1.umur = 34;
    manusia2.umur = 16;
    cout << "selisihnya adalah: " << manusia1.selisih_umur
            (manusia1, manusia2) << "!\n";
}
```

Disini kita membuat prototype dari fungsi yang akan kita buat. Kemudian, definisinya kita buat di luar struct dengan cara "memanggil" fungsi tersebut dengan tanda **"::"** (double colon = scope operator). Perhatikan penulisan:

```
int manusia::selisih_umur (manusia n, manusia m)
```

Perintah ini, mungkin, bisa diartikan: "fungsi `selisih_umur` yang mengambil dua argumen (`n` dan `m`) bertipe `manusia` yang merupakan bagian dari (struct) `manusia` yang mengembalikan nilai bertipe int" dan kemudian disusul definisi fungsi setelahnya.

BAB XI. Class

Class mirip dengan struct. Salah satu perbedaaannya adalah "sifat dasar" dari class adalah private. Private adalah jenis akses dari sebuah struktur data (seperti struct dan class). Pembagian tingkat akses ini dibagi menjadi 3 bagian:

- Public, member bisa langsung diakses dari luar oleh objectnya
- Private, member hanya bisa diakses dari member lainnya dalam class dan dari *friend*nya
- Protected, seperti private, tapi juga bisa di akses dari member dari turunan class tersebut

Untuk menentukan jenis – jenis akses ini dapat juga dibuat sendiri dengan menuliskan *keyword*nya, masing – masing sesuai dengan namanya. Default dari sebuah class adalah bersifat private. Contoh:

```
class my_class {
    int a;
    int b;
    void my_function (int arg1, int arg2) {
        //definisi fungsi
    }
};
```

Jika definisi dari class seperti di atas, maka **semua** membernya bersifat private. Bedakan dengan contoh di bawah ini:

```
class my_class {
    void my_function (int arg1, int arg2) {
```

```
        //definisi fungsi
    }
    public:
    int a;
    int b;
};
```

Definisi di atas bisa diartikan bahwa fungsi `my_function` bersifat private sedangkan variabel `a` dan `b` yang bersifat public.

Constructor

Kegunaan dari constructor adalah untuk melaksanakan hal – hal tertentu ketika object dari sebuah class dideklarasikan. Nama dari constructor harus sama dengan nama classnya.

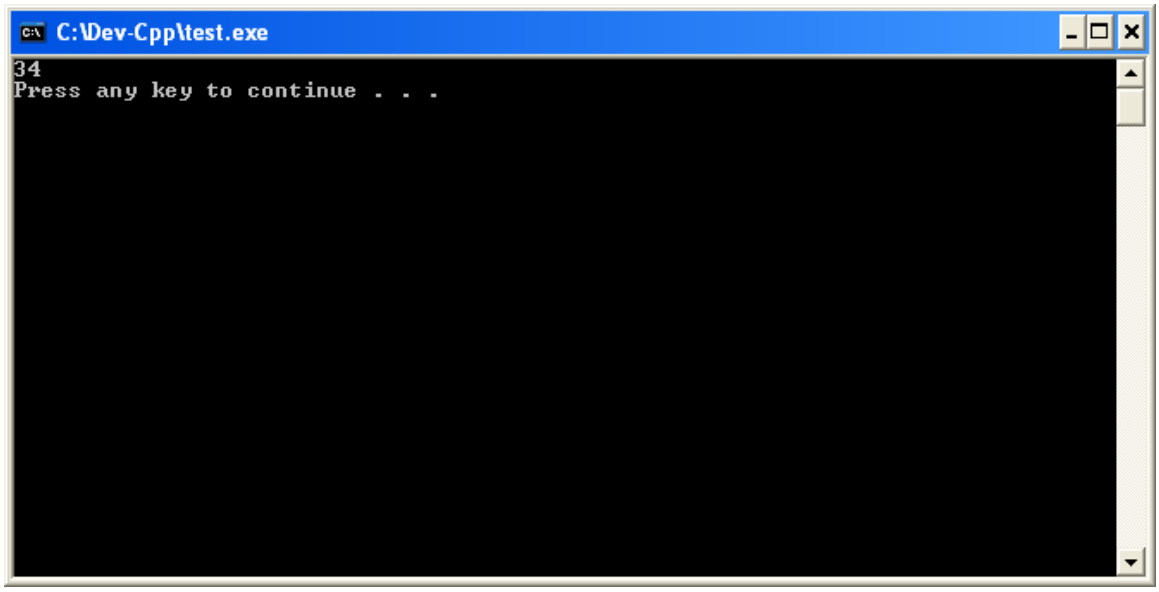
Destructor

Destructor adalah kebalikan dari constructor. Kalau constructor dipanggil pada waktu object dari suatu class dideklarasikan, maka destructor dipanggil ketika object dari class tersebut "dihancurkan", scope pada sebuah fungsi sudah berakhir (misalnya fungsi tempat object itu digunakan telah sampai pada batas scopenya). Destructor ditandai dengan tanda "~" (*tilde*) dan nama dari destructor harus sama dengan nama classnya.

Constructor dan destructor, tidak mengembalikan nilai apapun juga, bahkan `void`. Mengambil contoh dari kode yang telah kita gunakan di atas, maka contoh dari constructor dan destructor adalah:

```
#include <iostream>
using namespace std;
```

```
class my_class {
    int umur;
    public:
    my_class (int); //constructor
    int show_age() {
        return umur;
    }
    ~my_class(); //destructor

};

my_class::my_class (int age) {
     //definisi
     umur = age;
}

my_class::~my_class() {
    //definisi
    umur = 0;
}

void main () {
    my_class first(34);
    cout << first.show_age() << "\n";
}
```

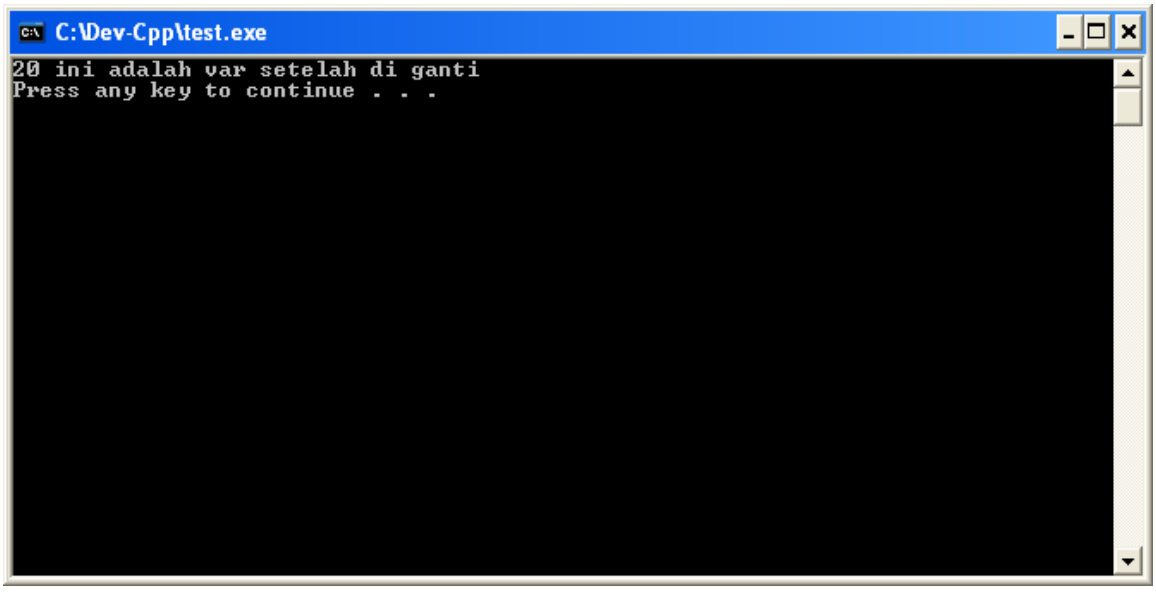


BAB XII. Pointer

Salah satu kegunaan dari pointer adalah sebagai "penunjuk" dari sebuah variabel. Jadi tanpa mengakses langsung variabel tersebut, anda dapat merubah isi dari variabel tersebut. Contoh:

```
int * pvar, var;
pvar = &var;
//cout << pvar << "\n";
//cout << &var << "\n";
*pvar = 20;
cout << var << "ini adalah nilai dari var setelah di ganti\n";
```

Output program tersebut adalah:

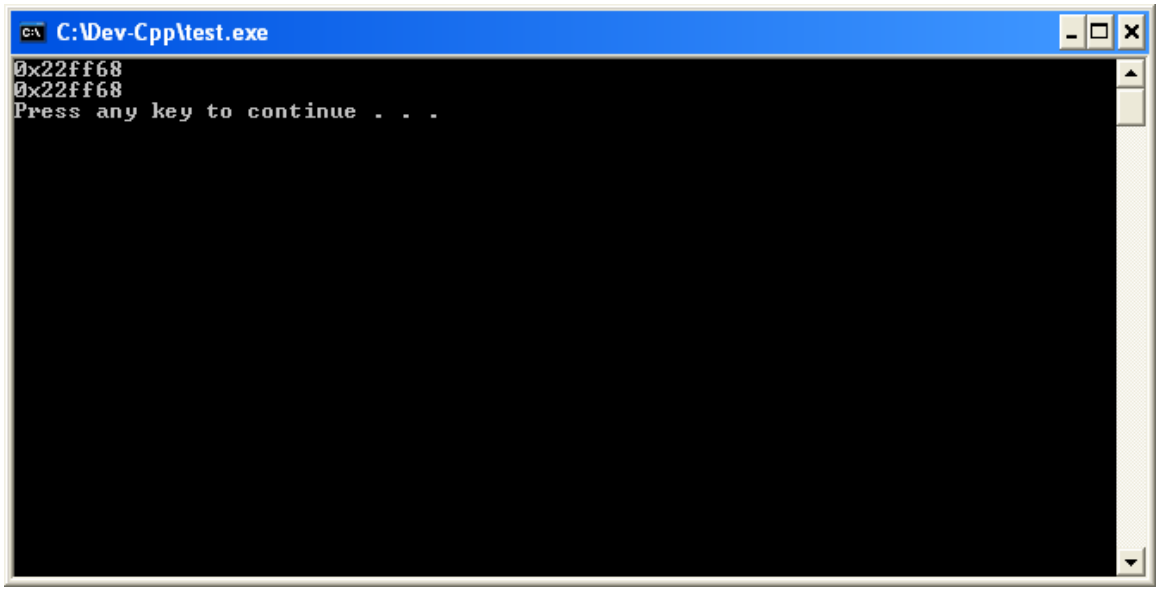


Variabel yang menjadi pointer dideklarasikan dengan memakai tanda *asterisk* ("*"). Pointer adalah variabel yang, biasanya, menunjuk ke sebuah variabel yang lain. Pointer menunjuk ke variabel lain dengan menunjuk ke alamat variabel tersebut (*by reference*). Dengan inisialisasi:

```
pvar = &var;
```

maka `pvar` akan menunjuk ke alamat `var`. Tanda "&" menandakan alamat dari variabel tersebut dalam memory komputer. Dimana inisialisasi di atas berarti `pvar` sama dengan alamat dari `var`. Kemudian dengan menggunakan tanda "*" untuk mengisi nilai, pointer tidak lagi menunjuk ke alamat dari `var` tapi langsung ke variabel itu sendiri / *dereference* (`*pvar = 20;`).

Untuk membuktikan bahwa `pvar` dan `&var` memiliki hasil yang sama, komentar di atas bisa di-*uncomment*, dan hasilnya adalah seperti ini (tergantung alamat variabel tersebut pada komputer):



Dynamic Memory

Penggunaan pointer bisa beragam. Ada kalanya kita ingin membuat sebuah variabel tapi tidak untuk digunakan secara permanen dalam program. Dengan pointer kita bisa membuat variabel – variabel sementara, dimana kita dapat menggunakannya ketika diinginkan dan kemudian, ketika tidak diperlukan lagi, dihapus untuk menghemat penggunaan memory dan sumber daya dalam komputer. Hal ini dilakukan dengan perintah `new` dan `delete`.

new

Perintah ini digunakan untuk membuat sebuah alokasi memory baru dalam komputer. Syntak:

```
nama_pointer = new tipe_data;
nama_pointer = new [array] tipe_data;
```

Misalnya, variabel yang ingin dibuat bertipe char. Maka:

```
char *pchar;
pchar = new char;
/* atau pchar = new [4] char; <--- ini membuat pchar menjadi
berarray 4 */
```

delete

delete adalah kebalikan dari new. Dengan kata lain, delete, sesuai dengan namanya, adalah perintah untuk menghapus variabel. Ini mengakibatkan memory yang sudah terpakai dalam komputer "dilepas" lagi untuk efisiensi penggunaan memory komputer. Syntak:

```
delete nama_pointer;
delete []nama_pointer;
```

Pointer dari sebuah class

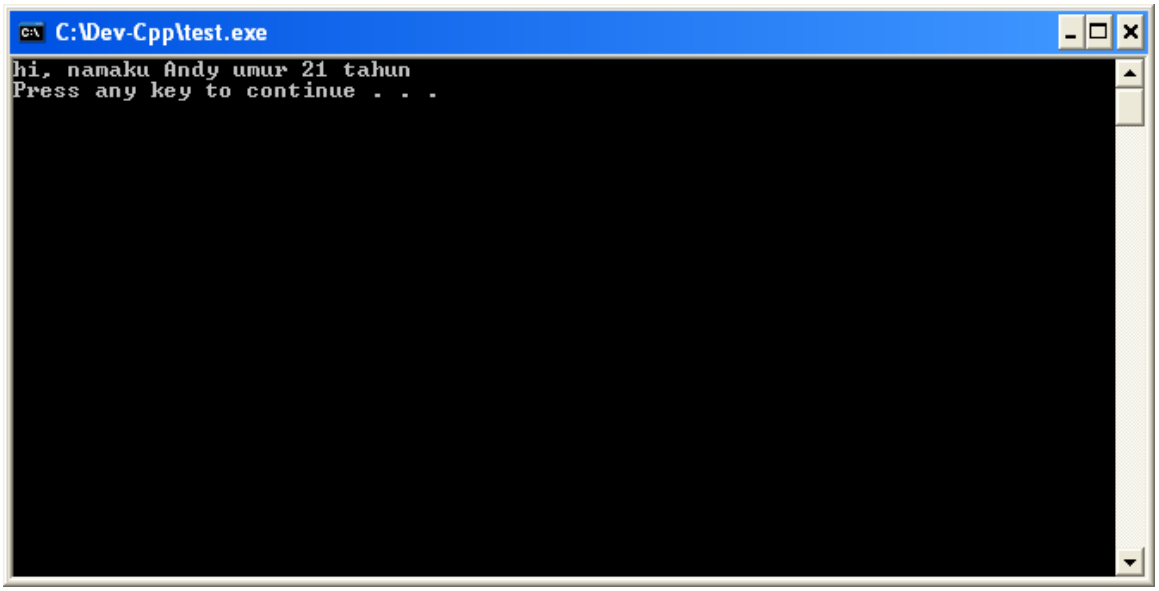
Jika adan ingin mendeklarasikan pointer dari sebuah class, maka syntak dari deklarasi pointer tersebut tidak jauh berbeda dengan ketika anda mendeklarasikan pointer seperti biasanya, tapi tipe data yang digunakan adalah nama class tersebut:

```
nama_class *nama_pointer;
```

Kembali lagi ke contoh *source code* sebelumnya, maka contohnya adalah sebagai berikut:

```
#include <iostream>
#include <string>
using namespace std;

class my_class {
    int umur;
    string nama;
    public:
    my_class (int, string);
    ~my_class ();
    int show_age () {
        return umur;
    }

    string show_name () {
        return nama;
    }
};

my_class::my_class(int age, string name) {
```

```
    umur = age;
    nama = name;
}

my_class::~my_class() {
    umur = 0;
    nama = "";
}

void main () {
    my_class first (21, "Andy"), *pfirst;
    pfirst = &first;
    /* pointer menunjuk ke fungsi sebuah variabel dari class dengan
       tanda "->" */
    cout << "hi, namaku " << pfirst->show_name() << " umur "
         << pfirst->show_age() << " tahun\n";
}
```



Referensi

http://cplusplus.com/tutorial

Biografi Penulis

Wirman a.k.a Chipp.

Adalah salah satu alumni S1 Universitas Katolik Atma Jaya Makassar jurusan Ekonomi Manajemen yang senang *ngutak-ngatik* komputer, terutama *software*. Sekarang bekerja dalam bidang yang sama sekali ***jauh*** dari dunia komputer, yaitu pelayaran (sebagai staff *finance*). Tidak mengaku *expert* (karena memang masih belajar), dan terus belajar untuk menjadi lebih baik.

Email: anthonio_wirman@yahoo.com

Blog: http://cyberianzone.blogspot.com

http://just-wirman.blogspot.com