# Chapter 6 – Class dan Object

## Pemrograman Berorientasi Obyek

Bahasa C++ memang berbeda dengan bahasa C. Bahasa C++ memiliki keunggulan dan perubahan yang besar dibandingkan dengan bahasa C. Salah satu perubahan mendasarnya adalah bahasa C++ dibuat untuk mendukung pemrograman berorientasi obyek.

Program adalah kumpulan instruksi yang disusun dengan urutan nalar yang tepat untuk menyelesaikan suatu persoalan. Dalam pembuatan program, pemrogram mempunyai cara pandang terhadap eksekusi sebuah program yang disebut sebagai paradigma pemrograman. Sebagai contoh dalam paradigma pemrograman berorientasi objek (OOP), pemrogram bisa melihat bahwa sebuah program adalah kumpulan objek yang saling berinteraksi, sedangkan dalam paradigma pemrograman terstruktur, pemrogram melihat bahwa sebuah program adalah suatu urutan instruksi yang dieksekusi secara berurutan.

Pemrograman Berorientasi Objek merupakan suatu paradigma pemrograman yang sudah sangat populer, meskipun metodologi pemrograman ini lebih baru dibandingkan dengan metodologi pemrograman terstruktur, tidak berarti pemrograman terstruktur harus ditinggalkan, karena sebenarnya secara internal pemrograman berorientasi objek juga dibangun dengan teknik pemrograman terstruktur, selain itu logika pemanipulasian objek kadang-kadang diekspresikan dengan pemrograman terstruktur juga. Pemrograman berorientasi obyek memiliki kelebihan, yaitu:

- Membuat suatu representasi teknis sedekat mungkin dengan pandangan konseptual dari dunia nyata.
- membuat kerangka analisis dan spesifikasi yang stabil.
- Memudahkan pengembangan dan perubahan programmer.

Tanpa kita sadari, dunia ini penuh dengan obyek. Obyek yang ada misalnya: sepeda, matahari, rumah, orang, anjing, topi, meja, dan masih banyak lagi. Ciri khas dari masing-masing obyek yang ada adalah dapat dilihat dan dapat digunakan. Setiap obyek pada dunia nyata juga memiliki ciri khas yang membedakannya dengan obyek lain terutama yang berbeda jenis. Sebagai contoh, obyek meja tulis, meja makan, dan meja belajar memiliki ciri yang sama, yaitu memiliki berat, memiliki kaki meja, memiliki warna meja, dan lain-lain. Kemudian obyek-obyek meja tersebut juga dapat menerima dan dilakukan operasi/kegiatan terhadapnya, misalnya kegiatan mengubah warna meja, kegiatan memotong atau menambah kaki meja dan lain-lain. Jadi sebuah obyek memiliki sifat yang melekat padanya dan memiliki hal yang dapat dikenakan atau dilakukannya. Antara obyek meja dengan obyek mobil memiliki perbedaan yang sangat besar.

Pemrograman berorientasi objek adalah suatu cara yang dipakai untuk mengorganisasikan program

kedalam suatu komponen logis (kelas), yang pada saat akan digunakan harus diinstansiasi menjadi sebuah objek (yaitu sebuah instan/ instance dari sebuah kelas). Sebuah kelas mempunyai anggota (data) dan metoda (fungsi yang bekerja untuk data tersebut), dengan kata lain Pemrograman Berorientasi Objek mengemas **data** (variabel / data member) dan **prosedur** (fungsi / function member / method) dalam sebuah objek sehingga kode program menjadi lebih fleksibel dan mudah dipelihara. Dalam Pemrograman Berorientasi Objek, objek yang dibuat melakukan suatu proses terhadap masukan tertentu dan mengeluarkan hasil tertentu dan pemakai tidak melihat bagaimana cara objek tersebut melakukan proses (karena program dibungkus di dalam objek).

**Kelas**

Kelas adalah *Blue Print* dari objek, yaitu *prototype* yang mendefinisikan variabel-variabel dan metoda-metoda (sub program) secara umum. Untuk dapat digunakan, suatu kelas harus **diinstansiasi** menjadi objek. Setiap objek yang diinstansiasi dari kelas yang sama akan mempunyai sifat dan tingkah laku yang sama.

Secara umum, ada dua bagian utama pada sebuah class C++, yaitu class declaration dan class body. Deklarasi kelas mendefinisikan nama class dan attributnya, sedangkan class body mendeklarasikan variabel dan method.

**Object**

Objek adalah suatu pengenal (***identifier***) yang menyatukan atribut (sering juga disebut ***property*** atau ***state***) dengan tingkah laku (yang disebut ***behaviour*** atau ***method***). Penyatuan *State* dan *Behaviour* ini pada konsep Pemrograman Berorientasi Objek disebut dengan istilah enkapsulasi (*encapsulation*).

Object mempunyai dua karakteristik, yaitu :

- Memiliki **attribut**, sebagai **status (*keadaan)***, yang kemudain disebut state.
- Memiliki tingkah laku, yang disebut ***behaviour***.

Contoh:

**Object Sepeda**

- Object Sepeda memiliki attribut (*state*) : pedal, roda, jeruji, warna, stang, jumlah roda.
- Object Sepeda memiliki tingkah laku (*behaviour*): kecepatan menaik, kecepatan menurun, memberhentikan, menjalankan, mengganti gigi.

## Class dan Object

Sering ada pertanyaan, class dan obyek duluan yang mana? Padahal kenyataannya: class adalah **blueprint/prototype** saja. Class merupakan **definisi** tentang state dan behaviour suatu objek. Bisa juga disebut class adalah **kumpulan** object yang memiliki **atribut** dan **service** yang **sama.** Sedangkan object adalah "**barang nyata**" dari sebuah class.

Contoh **class**: manusia sedangkan **object-nya** adalah kita, misalnya: Anton, Rudi, dan Amir.

Class memiliki sifat pewarisan, yang berarti sifat dari satu class dapat diturunkan ke class lain. Contoh pewarisan adalah class **dosen** memiliki semua atribut/state dan services dari class **manusia**. Dari contoh di atas, dapat dikatakan bahwa class dosen **mewarisi** semua atribut dan service dari class manusia. Class manusia adalah class **induk**, class dosen adalah class **anak**.

## Pembuatan Class pada C++

Class pada C++ bisa dianggap sebagai tipe data baru. Selain tipe data yang sudah dibawakan oleh C++, kita juga dapat membuat tipe data baru. Jika pada bahasa C/C++ tipe data baru dibuat menggunakan struct, pada C++ tipe data baru bisa dibuat dengan menggunakan class. Konsep ini merupakan konsep berorientasi obyek.

Struktur sederhana sebuah class pada C++:

```
class <namaclass>{
        //bagian member variabel / property class
        <tipedata> <namavariabel>;
        //bagian member function / method
        <tipedata> <namafunction>(<parameter>){
                //isi program dalam fungsi
        }
};
```

Contoh class:

```
class Kucing{
        //bagian member variabel yang bersifat private
        private:
        int umur;
        int berat;
        string jenis_kelamin;
        string nama_kucing;
        //bagian member function/method yang bersifat public
        public:
        void Bersuara();
        int tampilkanUmur();
}
```

**TIPS**

- Untuk membuat nama class, biasakanlah menggunakan huruf besar. Contohnya: Kucing, Rumah, Handphone, dan lain-lain.
- Untuk membuat nama variabel biasakanlah menggunakan nama yang mewakili property yang dimiliki dan melekat pada nama kelasnya. Pada contoh diatas, class Kucing memiliki berat, umur, dan nama yang melekat erat padanya. Setiap kucing yang ada didunia ini pada umumnya memiliki berat, umur, dan nama yang berbeda-beda satu sama lain. Untuk menamai member variabel, jika nama variabel hanya terdiri dari satu kata gunakanlah huruf kecil, sedangkan jika terdiri dari lebih dari satu kata, gunakan huruf kecil pada huruf kata pertama, sedangkan untuk

kata selanjutnya gunakan huruf besar pada huruf-huruf pertamanya. Contoh: string namaKucing, float ipkMahasiswa, int berat, dan lain-lain.

- Untuk membuat nama member function, gunakanlah cara penulisan yang tepat untuk menggambarkan secara benar setiap nama function yang ada. Biasakanlah memberi nama function sesuai dengan behaviour yang memang dikerjakannya. Contohnya: void cariData(string judul), atau int ambilNilai() dan lain-lain.

## Mendefinisikan Obyek

Setelah kita selesai membuat class baru, maka kita bisa menggunakan class tersebut adalah dengan menginisialisasinya (dengan membuat sebuah atau beberapa obyek) dari class tersebut. Membuat obyek bisa dianggap seperti membuat variabel yang bertipe class yang kita buat. Contoh:

```
Kucing kucingku;
Orang anton;
Mobil kijang;
```

Class dan obyek adalah berbeda. Class merupakan template dari member variabel dan member function yang dapat dibuat wujud nyatanya dalam sebuah obyek. Pada contoh diatas, Kucing adalah class. Kucing tidak bisa langsung digunakan dalam program. Untuk bisa menggunakan Kucing, yang harus dilakukan adalah membuat obyeknya, yaitu kucingku. Pada dunia nyata kucingku bisa disebut sesuai dengan nama kucing yang kia pelihara. Jadi contoh diatas mungkin bisa diubah menjadi Kucing katty. Dimana katty adalah obyek dari class Kucing yang dapat digunakan dalam program.

## Mengakses Member Variabel

Setelah kita membuat obyek seperti:

```
Kucing katty;
```

Cara mengakses member variabel adalah dengan menggunakan tanda titik (.). Contoh jika kita hendak mengisi data berat badan katty dengan nilai 8 kg, maka yang harus dilakukan adalah:

```
Katty.berat = 8;
```

Demikian juga dengan nama, umur, dan jenis kelamin.

```
Katty.nama = “Katty”;
Katty.jenis_kelamin = “jantan”;
Katty.umur = 2;
```

## Mengakses Member Function/Method

Sedangkan cara untuk mengakses member function dari suatu class adalah dengan juga menggunakan tanda titik pada obyeknya. Contoh:

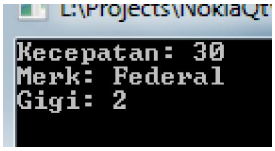
```
katty.Bersuara();
katty.tampilkanUmur();
```

## Labs 1. Pembuatan class Sepeda

Buatlah project baru dan tulis kode berikut:

```
#include <QtCore/QCoreApplication>
#include <iostream>
using namespace std;
class Sepeda{
private:
    int kecepatan;
    int gigi;
    string merk;
public:
    void ubahKecepatan(int kec);
    void ubahGigi(int g);
    void setMerk(string m);
    void tampilSepeda();
};
void Sepeda::ubahKecepatan(int kec){
    this->kecepatan = kec;
}
void Sepeda::ubahGigi(int g){
    this->gigi = g;
}
void Sepeda::setMerk(string m){
    this->merk = m;
}
void Sepeda::tampilSepeda(){
    cout <<"Kecepatan: "<<this->kecepatan<<endl<<
           "Merk: "<<this->merk<<endl<<
           "Gigi: "<<this->gigi;
}
int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    Sepeda objSpd;
    objSpd.ubahGigi(2);
    objSpd.ubahKecepatan(30);
    objSpd.setMerk("Federal");
    objSpd.tampilSepeda();
    return a.exec();
}
```

Hasil:




**Analisa**

- Program diatas membuat sebuah class bernama Sepeda. Di dalam class Sepeda, terdapat dua

bagian, bagian pertama berisi semua member variabel yang bersifat private, yaitu kecepatan, gigi, dan merk. Pada bagian kedua terdapat member function yang hanya beisi judul method saja sedangkan implementasinya diletakkan diluar class Sepeda.

- Diluar kelas Sepeda, kita mendefinisikan semua implementasi method dari semua member function yang sudah kita definisikan diatas. Untuk mengakses member function dari luar kelasnya, digunakan tanda :: setelah nama class. Implementasi method bisa menggunakan cara lain yang akan dijelaskan dibagian-bagian berikutnya.
- Pada function main, kita membuat obyek dari class Sepeda yang bernama objSpd dan kemudian kita akses semua member functionnya.
- Sebelum menampilkan hasil kita isi terlebih dahulu kecepatan, gigi, dan merk dari Sepeda yang kita buat.
- Keyword this mengacu pada class itu sendiri (class Sepeda) dan merupakan variabel pointer. Keyword tersebut digunakan untuk mengakses semua member variabel dan member method class Sepeda.

**TIPS**

Keyword this pada class Sepeda merupakan kata kunci untuk mengakses class yang didefinisikan (kelas dirinya sendiri). Tanda -> merupakan tanda bahwa this merupakan obyek pointer. Cara lain untuk mengakses kelas itu sendiri adalah dengan menggunakan <namakelas>:: diikuti nama method / variabel member .

Contoh :

```
void Sepeda::ubahKecepatan(int kec){
    Sepeda::kecepatan = kec;
}
```

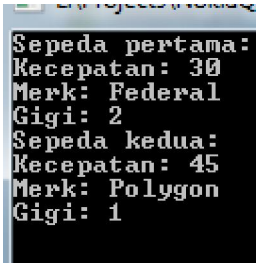
### Labs 2. Pembuatan obyek Sepeda.

Buatlah project baru dan tulis kode berikut:

```
#include <QtCore/QCoreApplication>
#include <iostream>
using namespace std;
class Sepeda{
private:
    int kecepatan;
    int gigi;
    string merk;
public:
    void ubahKecepatan(int kec);
    void ubahGigi(int g);
    void setMerk(string m);
    void tampilSepeda();
};
void Sepeda::ubahKecepatan(int kec){
    this->kecepatan = kec;
}
void Sepeda::ubahGigi(int g){
    this->gigi = g;
```

```
}
void Sepeda::setMerk(string m){
    this->merk = m;
}
void Sepeda::tampilSepeda(){
    cout <<"Kecepatan: "<<this->kecepatan<<endl<<
           "Merk: "<<this->merk<<endl<<
           "Gigi: "<<this->gigi;
}
int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    cout<<"Sepeda pertama:\n";
    Sepeda objSpd;
    objSpd.ubahGigi(2);
    objSpd.ubahKecepatan(30);
    objSpd.setMerk("Federal");
    objSpd.tampilSepeda();
    cout<<"\nSepeda kedua:\n";
    Sepeda objSpd2;
    objSpd2.ubahGigi(1);
    objSpd2.ubahKecepatan(45);
    objSpd2.setMerk("Polygon");
    objSpd2.tampilSepeda();
    return a.exec();
}
```

Hasil:




**Analisa**

- Program diatas merupakan pengembangan dari program sebelumnya dimana kita membuat satu lagi variabel objSpd2.
- Terlihat bahwa masing-masing obyek sepeda yang terbuat memiliki data yang berbeda-beda satu sama lain.
- Artinya class hanyalah merupakan template / blueprint saja, dimana data-data dan tingkah laku dari kelas haruslah dilakukan oleh obyeknya. Jadi obyek adalah bentuk nyata dari sebuah kelas yang memiliki data dan method yang berbeda-beda satu sama lain.

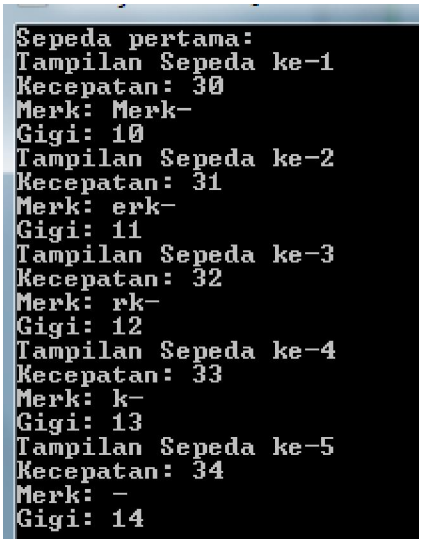
## Labs 3. Pembuatan Obyek Array Sepeda

Buatlah program beikut ini:

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

```
using namespace std;
class Sepeda{
private:
    int kecepatan;
    int gigi;
    string merk;
public:
    void ubahKecepatan(int kec);
    void ubahGigi(int g);
    void setMerk(string m);
    void tampilSepeda();
};
void Sepeda::ubahKecepatan(int kec){
    this->kecepatan = kec;
}
void Sepeda::ubahGigi(int g){
    this->gigi = g;
}
void Sepeda::setMerk(string m){
    this->merk = m;
}
void Sepeda::tampilSepeda(){
    cout <<"Kecepatan: "<<this->kecepatan<<endl<<
           "Merk: "<<this->merk<<endl<<
           "Gigi: "<<this->gigi;
}
int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    cout<<"Sepeda pertama:\n";
    Sepeda objSpdArray[5];
    for(int i=0;i<5;i++){
        objSpdArray[i].setMerk("Merk-"+i);
        objSpdArray[i].ubahGigi(i+10);
        objSpdArray[i].ubahKecepatan(i+30);
    }
    for(int i=0;i<5;i++){
        cout<<"Tampilan Sepeda ke-"<<(i+1)<<endl;
        objSpdArray[i].tampilSepeda();
        cout<<endl;
    }
    return a.exec();
}
```

Hasil:




**Analisa:**

- Program diatas merupakan pengembangan lagi dari labs 1.
- Program diatas membuat obyek dari class Sepeda dalam bentuk Array 1 dimensi yang bertipe Sepeda.
- Array yang bertipe class Sepeda tersebut tetap memiliki indeks dari 0 sampai dengan n-1
- Masing-masing obyek elemen array objSpdArray berisi data-data yang berbeda-beda satu sama lainnya.

**TIPS**

Kita juga dapat melakukan assigment / penugasan terhadap obyek ke obyek lain. Contoh kita memiliki class Sepeda dan kita membuat obyek spd1 dan spd2.

```
Sepeda spd1,spd2;
spd1.setMerk("X");
spd1.ubahKecepatan(50);
spd1.ubahGigi(4);
```

maka bisa dilakukan:

```
spd2 = spd1;
```

Jika spd2 ditampilkan, dengan sp2.tampilSepeda(), maka nilai yang ditampilkan akan sama persis dengan **nilai spd1.**

Class yang kita definisikan memiliki member method. Semua member method tersebut dapat kita gunakan. Apa yang kita buat dalam member method akan membuat kompiler mendaftarkan semua method yang kita buat kedalam memory sehingga hanya method yang kita daftarkan saja yang bisa kita akses dari class kita.

## Hak Akses Member Variabel dan Method Variabel

Pada pemrograman berorientasi obyek, terdapat konsep penting yang bernama enkapsulasi. Konsep tersebut berarti kita "membungkus" semua member variabel dan member method kedalam suatu class termasuk hak akses terhadap mereka. Apa arti hak akses? Hak akses adalah bagaimana class yang terenkapsulasi tersebut "menyembunyikan" hal-hal yang tidak perlu / tidak boleh dilihat dari luar class. Dengan adanya hak akses tersebut semua data dan method akan terlindungi dan tidak termodifikasi.

Kita dapat analogikan dengan kasus nyata sebuah benda, misalnya AC. AC merupakan alat elektronik yang rumit dan mampu mendinginkan ruangan. Jika seseorang memasang AC maka AC akan melindungi dirinya dengan hak akses. Kita sebagai orang awam tentang AC hanya diperbolehkan mengakses yang diperbolehkan saja untuk mengantisipasi hal-hal yang tidak diinginkan seperti misalnya AC akan rusak. Kita hanya diberikan tombol-tombol sederhana dan mungkin remote untuk mengatur semua tentang AC, kita tidak bisa mengakses hardwarenya, kabel didalamnya, PCB nya, kondensatornya dan lain-lain. Yang bisa melakukan itu adalah para ahli AC. Dengan demikian AC sudah berusaha melindungi dirinya dari tangan-tangan orang awam yang memang tidak berhak.

Demikian pula pada class, class juga memiliki dua bagian: member variabel dan member method. Kedua bagian ini berbeda fungsinya. Member variabel digunakan untuk menyimpan sifat-sifat yang dimiliki dan melekat pada class sedangkan method digunakan untuk melakukan operasi/kegiatan terhadap class tersebut. Jika kita bandingan dengan AC, maka method bisa dibilang sebagai remote/tombol. Sehingga untuk mengakses data-data yang ada pada class kita sangat direkomendasikan untuk menggunakan method bukan mengaksesnya secara langsung.

Class pada C++ memiliki cara melindungi dirinya yaitu dengan menggunakan keyword private, protected dan public. Keyword private atau protected biasanya digunakan pada semua variabel member sedangkan keyword protected atau public digunakan pada semua variabel method. Dengan menggunakan keyword private, maka bagian private tersebut tidak akan bisa diakses dari luar class, harus dari dalam class tersebut atau berada dalam method class tersebut, sedangkan jika public maka bisa akses dari luar class.

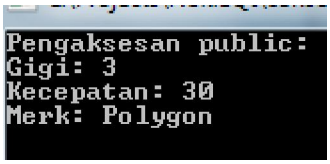
### Labs 4. Perbedaan private dan public pada member variabel

Buatlah program berikut:

```
#include <QtCore/QCoreApplication>
#include <iostream>
using namespace std;
class Sepeda{
private:
    int kecepatan;
    int gigi;
    string merk;
```

```
public:
    int pkecepatan;
    int pgigi;
    string pmerk;
};
int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    cout<<"Pengaksesan public:\n";
    Sepeda s;
    s.pgigi = 3;
    s.pkecepatan = 30;
    s.pmerk = "Polygon";
    cout<<"Gigi: "<<s.pgigi<<endl;
    cout<<"Kecepatan: "<<s.pkecepatan<<endl;
    cout<<"Merk: "<<s.pmerk<<endl;
    return a.exec();
}
```

Hasil:




**Analisa**

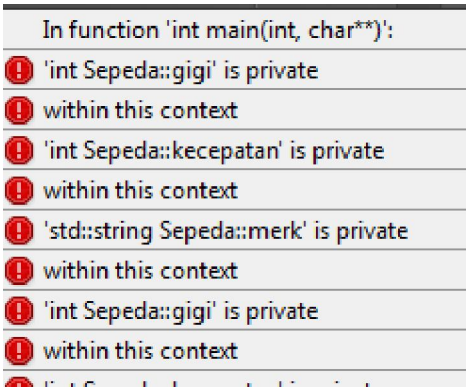
- Semua variabel member yang bersifat public dapat diakses dan diisi dengan baik dari luar class, dalam hal ini adalah function int main(). Function int main() berada diluar class Sepeda

Akan terjadi hal yang berbeda jika kita mengakses semua variabel member yang bersifat private.
Ubahlah program pada labs 5 diatas menjadi seperti berikut ini:

```
#include <QtCore/QCoreApplication>
#include <iostream>
using namespace std;
class Sepeda{
private:
    int kecepatan;
    int gigi;
    string merk;
public:
    int pkecepatan;
    int pgigi;
    string pmerk;
};
int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    cout<<"Pengaksesan public:\n";
    Sepeda s;
    s.gigi = 3;
    s.kecepatan = 30;
```

```
    s.merk = "Federal";
    cout<<"Gigi: "<<s.gigi<<endl;
    cout<<"Kecepatan: "<<s.kecepatan<<endl;
    cout<<"Merk: "<<s.merk<<endl;
    return a.exec();
}
```

Hasil:




**Analisa:**

- Akan terjadi compile time error, karena kita mengakses variabel member yang bersifat private. Berarti class Sepeda sudah bisa menerapkan fungsi enkapsulasi dan melindungi data-datanya dari pengaksesan langsung.

## Member Function / Member Method

Seperti yang sudah dijelaskan, member method merupakan bagian yang harus dideklarasikan sebagai bagian public. Salah satu kegunaan member function adalah mengakses semua member variabel dan tetap mendukung enkapsulasi. Cara untuk membuat member method adalah dengan mendeklarasikannya pada bagian public, sedangkan implementasi kodingnya berada diluar kelas. Berikut adalah contohya.

```
class Sepeda{
private:
    int kecepatan;
    int gigi;
    string merk;
public:
    void setKecepatan(int k);
    void setGigi(int g);
    void setMerk(string m);
};
```

Di dalam pemrograman beriorientasi obyek pada umumnya member function minimal selalu mewakili semua member variabelnya. Misal kita memiliki 1 buah member variabel bernama umur, maka minimal kita akan memiliki satu buah member function, misalnya bernama ubahUmur(int u).

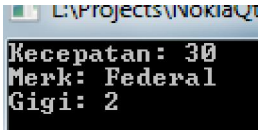
## Labs 5. Member function dan implementasinya

Buatlah program berikut ini:

```
#include <QtCore/QCoreApplication>
#include <iostream>
using namespace std;
//pembuatan class Sepeda
class Sepeda{
private:
    //daftar member variabel
    int kecepatan;
    int gigi;
    string merk;
public:
    //daftar member function
    void ubahKecepatan(int kec);
    void ubahGigi(int g);
    void setMerk(string m);
    void tampilSepeda();
};
//implementasi member function berada diluar class Sepeda
//function ubahKecepatan menerima input jumlah kecepatan
//mengubah kecepatan Sepeda
void Sepeda::ubahKecepatan(int kec){
    this->kecepatan = kec;
}
//function ubahGigi menerima input jumlah gigi
//mengubah gigi Sepeda
void Sepeda::ubahGigi(int g){
    this->gigi = g;
}
//function setMerk menerima input string merk
//mengisi merk Sepeda
void Sepeda::setMerk(string m){
    this->merk = m;
}
//function tampilSepeda tidak menerima input
//fungsinya hanya untuk menampilkan informasi obyek Sepeda
void Sepeda::tampilSepeda(){
    cout <<"Kecepatan: "<<this->kecepatan<<endl<<
           "Merk: "<<this->merk<<endl<<
           "Gigi: "<<this->gigi;
}
//function main
int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    Sepeda objSpd;
    objSpd.ubahGigi(2);
    objSpd.ubahKecepatan(30);
    objSpd.setMerk("Federal");
    objSpd.tampilSepeda();
    return a.exec();
```

```
}
```

Hasil:




**Analisa**

- Semua member variabel yang dimiliki tidak diakses secara langsung dari function main, tapi melalui method-methodnya.
- Pada program diatas setiap member variabel memiliki minimal satu buah method member
- Terdapat satu buah method tambahan yang berfungsi untuk menampilkan semua informasi mengenai sepeda
- Setiap method member dapat menerima input dan mengeluarkan output.
- Kata kunci this-> pada method member berfungsi untuk mengakses semua member variabel yang terdapat pada class Sepeda yang biasanya bersifat private.
- Implementasi method member berada diluar class Sepeda dan dimulai dengan nama classnya kemudian diikuti tanda :: yang artinya mengakses member method.

## Accessor dan Mutator Method

Pada pemrograman berorientasi obyek dengan C++, kita memiliki method member. Tujuan dari method member selain memberi tingkah laku dari class tersebut adalah melakukan akses terhadap semua member variabel yang bersifat private agar tetap bisa diakses dari luar class.

Member method yang berkaitan dengan member variabel ada 2 jenis, yaitu member method yang berfungsi untuk mengeset / mengisi nilai member variabel dan member method yang berfungsi untuk mengambil nilai member variabel.

1. Accessor method: method ini berfungsi untuk mengambil nilai dari sebuah member variabel.
   Asesor method biasanya dinamai :
   **<tipedataMemberVariabel> get<NamaMemberVariabel>();**
   Contoh:
   **int getUmur();**
2. Mutator method: method ini berfungsi untuk mengisi / mengeset nilai kepada sebuah member variabel
   Mutator method biasanya dinamai :
   **void set<NamaMemberVariabel>(<tipedataMemberVariabel> <namavariabel>);**
   Contoh:
   **void setUmur(int u);**

## Labs 6. Penggunaan accessor dan mutator method

Tulislah program berikut ini:

```
#include <QtCore/QCoreApplication>
#include <iostream>
using namespace std;
class Kucing{
private:
    int umur;
    float berat;
    string nama;
public:
    //asesor method
    int getUmur();
    float getBerat();
    string getNama();
    //mutator method
    void setUmur(int u);
    void setBerat(float b);
    void setNama(string s);
    //method tambahan
    void berlari();
};
//implementasi
int Kucing::getUmur(){
    return this->umur;
}
float Kucing::getBerat(){
    return this->berat;
}
string Kucing::getNama(){
    return this->nama;
}
void Kucing::setUmur(int u){
    this->umur = u;
}
void Kucing::setBerat(float b){
    this->berat = b;
}
void Kucing::setNama(string s){
    this->nama = s;
}
void Kucing::berlari(){
    cout<<"Kucing "<<this->getNama()<<" sedang berlari!";
}
int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    Kucing mycat;
    mycat.setNama("Katty");;
    mycat.setBerat(4);
    mycat.setUmur(2);
    cout<<"Kucingku bernama "<<mycat.getNama()<<", dia berbobot "<<
            mycat.getBerat()<<" kg dan sudah berumur "<<mycat.getUmur()
```

```
            <<" tahun sekarang."<<endl;
    mycat.berlari();
    return a.exec();
}
```

Hasil:

```
Kucingku bernama Katty, dia berbobot 4 kg dan sudah berumur 2 tahun sekarang.
Kucing Katty sedang berlari!
```

**Analisa**

- Program diatas memperlihatkan bagaimana setiap member variabel memiliki tepat dua buah method member, dimana setiap method member yang satu berfungsi sebagai asesor method dan yang lain berfungsi sebagai mutator method.
- Terdapat sebuah method tambahan yaitu berlari yang hendak menggambarkan bahwa selain asesor dan mutator kita masih diperbolehkan membuat method lainnya.
- Asesor method mengambil data member variabel sehingga dibuat fungsi berupa function non void, sedangkan mutator method mengeset data member variabel sehingga dibuat fungsi berupa function void yang menerima parameter yang sesuai dengan tipe data member variabelnya.

## Constructor dan Destructor

Kita dapat mendeklarasikan variabel biasa dan kemudian melakukan inisialisasi terhadap variabel tersebut dengan mudah. Contoh:

```
int umur = 5;
```

Inisialisasi variabel berfungsi untuk mengisi suatu nilai awal terhadap suatu variabel yang kita deklarasikan. Variabel tersebut masih bisa kita ubah-ubah lagi nilainya dikemudian waktu. Nah bagaimana untuk menginisialisasi variabel member pada suatu class? Caranya dengan membuat method yang berjenis constructor method. Sedangkan untuk mendealokasi dan melakukan finalisasi sebuah class kita gunakan destructor method. Constructor berfungsi untuk menginisialisasi obyek dari class dan mempersiapkan ruang memory, sedangkan destructor menghapus dan membersihkan obyek ketika sudah tidak terpakai dan membebaskan memory yang tadinya terpakai.

Constructor method merupakan method yang namanya **sama** dengan nama classnya dan bersifat **public** tapi **tidak** berjenis void ataupun non void. Constructor **dapat** menerima parameter namun **tidak bisa** mengembalikan nilai apapun.

Desktruktor method merupakan method kebalikan dari constructor yang juga bernama sama dengan nama classnya namun diawali dengan tanda **~**. Destructor **tidak boleh** memiliki parameter apapun.

Contoh jika kita memiliki class bernama Sepeda, maka kita dapat membuat constructor dengan nama Sepeda() juga. Sedangkan destructor method sama dengan constructor namun diawali dengan tanda ~ didepannya. Contoh:

```
class Sepeda{
      private:
      //member variabel
      public:
      //konstruktor
      Sepeda();
      //destruktor
      ~Sepeda();
};
```

## Default Constructor

Pada bahasa C++ semua class yang telah dibuat PASTI memiliki constructor walaupun tidak kita buat. Compiler bahasa C++ pasti membuatnya walau secara implisit. Constructor yang bernama sama dengan nama classnya dan tidak berparameter disebut default constructor. Secara default pasti semua class ada default constructornya. Kapan kita menggunakan constructor? Setiap kali kita membuat obyek baru (melakukan instansiasi obyek), maka kita memanggil constructor default.

Contoh:

```
Sepeda sepedaku;
```

Berarti kita memanggil default konstruktor bernama Sepeda() tanpa parameter apapun. Jika kita membuat konsruktor dengan menggunakan parameter seperti misalnya:

```
Sepeda(string merk, int berat);
```

Maka pada saat instansiasi kita menggunakan cara sebagai berikut:

```
Sepeda sepedaku("Federal",2);
```

Arti instasiasi diatas adalah kita memanggil konstruktor yang berparameter dua buah, string dan integer.

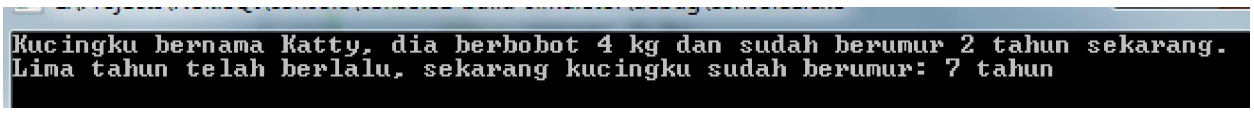
### Labs 7. Menggunakan Constructor dan Destructor

Buatlah program berikut:

```
#include <QtCore/QCoreApplication>
#include <iostream>
using namespace std;
class Kucing{
private:
    int umur;
    float berat;
    string nama;
public:
    //konstruktor
    Kucing(int umur);
    //desktruktor
    ~Kucing();
```

```
    //asesor method
    int getUmur();
    float getBerat();
    string getNama();
    //mutator method
    void setUmur(int u);
    void setBerat(float b);
    void setNama(string s);
};
//implementasi konstruktor dan desktruktor
Kucing::Kucing(int u){
    this->umur = u;
}
Kucing::~Kucing(){
}
//implementasi function
int Kucing::getUmur(){
    return this->umur;
}
float Kucing::getBerat(){
    return this->berat;
}
string Kucing::getNama(){
    return this->nama;
}
void Kucing::setUmur(int u){
    this->umur = u;
}
void Kucing::setBerat(float b){
    this->berat = b;
}
void Kucing::setNama(string s){
    this->nama = s;
}
int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    Kucing mycat(2);
    mycat.setNama("Katty");;
    mycat.setBerat(4);
    cout<<"Kucingku bernama "<<mycat.getNama()<<", dia berbobot "<<
            mycat.getBerat()<<" kg dan sudah berumur "<<mycat.getUmur()
            <<" tahun sekarang."<<endl;
    mycat.setUmur(7);
    cout<<"Lima tahun telah berlalu, sekarang kucingku sudah berumur:
"<<mycat.getUmur()<<" tahun";
    return a.exec();
}
```

Hasil:



**Analisa:**

- Program diatas menunjukkan pemakaian konstruktor dan desktruktor. Constructor digunakan untuk menginisialisasi umur kucing pada saat awal pertama obyek dibuat, kemudian pada akhirnya kita juga tetap dapat mengubah umur kucing dibagian akhir program.
- Destruktor yang kita buat merupakan default desktruktor dimana desktruktor tidak boleh memiliki parameter apapun.

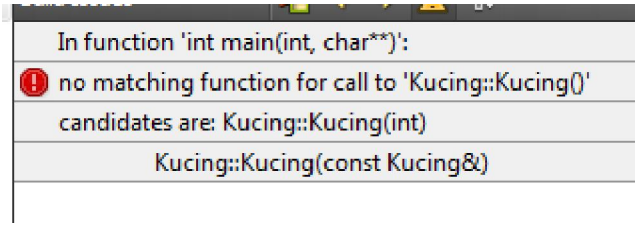
**TIPS**

Jika kita sudah membuat konstruktor yang memiliki parameter pada class kita, maka secara otomatis default constructor yang dibuat oleh compiler tidak ada lagi, sehingga ketika kita melakukan instansiasi pada class Kucing diatas tanpa parameter pasti akan error.

Contoh, tambahkan satu baris berikut ini pada bagian akhir kode pada labs 7 sebelum return a.exec().

```
Kucing kucingku2;
```

Ketika dilakukan kompilasi akan menghasilkan error sebagai berikut:




Error diatas mengatakan bahwa class Kucing tidak memiliki function yang bernama Kucing::Kucing(), yang artinya method constructor defaultnya sudah hilang. Agar kita dapat menggunakan baris Kucing kucingku2; maka kita harus menambah method constructor lagi yang tidak berparameter.

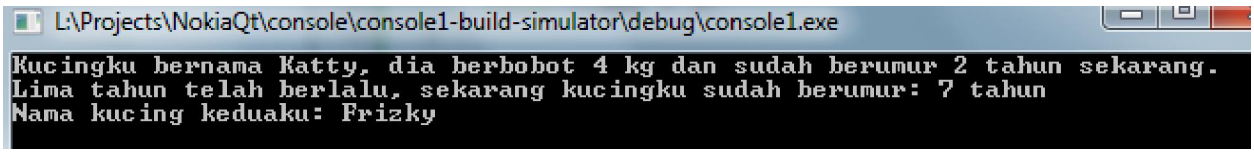
### Labs 8. Percobaan Menambah Constructor Method

Buatlah program berikut:

```
#include <QtCore/QCoreApplication>
#include <iostream>
using namespace std;
class Kucing{
private:
    int umur;
    float berat;
    string nama;
public:
    //konstruktor
    Kucing(int umur);
    Kucing();
    //desktruktor
    ~Kucing();
    //asesor method
    int getUmur();
    float getBerat();
    string getNama();
    //mutator method
```

```
    void setUmur(int u);
    void setBerat(float b);
    void setNama(string s);
};
//implementasi konstruktor dan desktruktor
Kucing::Kucing(int u){
    this->umur = u;
}
Kucing::Kucing(){
}
Kucing::~Kucing(){
    cout<<"Obyek sudah dihancurkan!";
}
//implementasi function
int Kucing::getUmur(){
    return this->umur;
}
float Kucing::getBerat(){
    return this->berat;
}
string Kucing::getNama(){
    return this->nama;
}
void Kucing::setUmur(int u){
    this->umur = u;
}
void Kucing::setBerat(float b){
    this->berat = b;
}
void Kucing::setNama(string s){
    this->nama = s;
}
int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    Kucing mycat(2);
    mycat.setNama("Katty");;
    mycat.setBerat(4);
    cout<<"Kucingku bernama "<<mycat.getNama()<<", dia berbobot "<<
            mycat.getBerat()<<" kg dan sudah berumur "<<mycat.getUmur()
            <<" tahun sekarang."<<endl;
    mycat.setUmur(7);
    cout<<"Lima tahun telah berlalu, sekarang kucingku sudah berumur:
"<<mycat.getUmur()<<" tahun"<<endl;
    Kucing kucingku2;
    kucingku2.setNama("Frizky");
    cout<<"Nama kucing keduaku: "<<kucingku2.getNama();
    return a.exec();
}
```

Hasil:




**Analisa:**

- Pada program C++, kita dapat membuat konstruktor method lebih dari satu, asal tidak sama. Konsep diatas dinamakan dengan polymorfisme (OVERLOADING) yang akan dibahas lebih lanjut dibab-bab berikutnya.
- Dengan mendefinisikan konstruktor tanpa parameter maka kita dapat menginstansiasi obyek dengan cara biasa, seperti pada contoh Kucing kucingku2;

## Constructor Dengan nilai Default

Constructor dapat memiliki nilai default sehingga jika konstruktor yang dipanggil tidak diisi nilai, maka nilai-nilai lainnya akan tetap diinisialisasi dengan nilai defaultnya. Hal ini diperlukan untuk mempermudah menginisialisasi data variabel member. Penggunaan nilai default ini juga memungkinkan kita untuk tidak memasukkan semua parameter pada pemanggilan konstruktor.

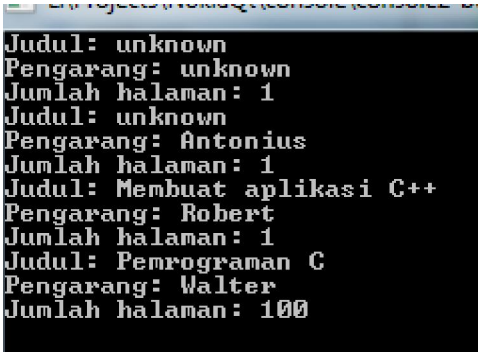
### Labs. Penggunaan Constructor dengan Nilai Default

Buatlah program berikut:

```
#include <QtCore/QCoreApplication>
#include <iostream>
using namespace std;
class Buku{
private:
    int jmlhal;
    string pengarang;
    string judul;
public:
    Buku(string pengarang="unknown", string judul="unknown",int jmlhal=1){
        Buku::jmlhal = jmlhal;
        Buku::pengarang = pengarang;
        Buku::judul = judul;
    }
    void tampilInfo(){
        cout<<"Judul: "<<Buku::judul<<endl;
        cout<<"Pengarang: "<<Buku::pengarang<<endl;
        cout<<"Jumlah halaman: "<<Buku::jmlhal<<endl;
    }
};
int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    Buku b1;
    Buku b2("Antonius");
    Buku b3("Robert","Membuat aplikasi C++");
    Buku b4("Walter","Pemrograman C",100);
```

```
    b1.tampilInfo();
    b2.tampilInfo();
    b3.tampilInfo();
    b4.tampilInfo();
    return a.exec();
}
```

Hasil:




**Analisa:**

- Program diatas menunukkan bahwa kita dapat membuat konstruktor dengan nilai default, yaitu dengan menggunakan parameter dan langsung diinisialisasi dengan menggunakan tanda sama dengan (=).
- Pada pemanggilan konstruktor, terlihat bahwa jika konstruktor tidak diisi parameter apapun maka ketika data ditampilkan semua isi member variabel sesuai dengan nilai defaultnya.
- Pada pemanggilan konstruktor kedua, yaitu dengan satu parameter string, maka string tersebut mengacu pada parameter pertama, yaitu Pengarang, sehingga judul dan jumlah halaman berisi nilai default.
- Pada pemanggilan konstruktor ketiga, yaitu dengan dua parameter string, maka kedua parameter itu mengisi pengarang dan judulnya (hal ini sesuai dengan urutan penempatan pada pendefinisian method konstruktor pada program), sedangkan variabel member lain berisi default
- Pada pemanggilan ketiga, ketiga parameter diisi sehingga semua nilai default berubah.

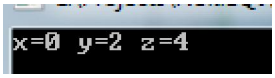
TIPS

Kita juga dapat memberi nilai default dengan cara lain, perhatikan contoh berikut:

```
#include <QtCore/QCoreApplication>
#include <iostream>
using namespace std;
class Contoh{
private:
    int x;
    int y;
    int z;
public:
    Contoh():x(0),y(2),z(4){
    }
```

```
    void tampilInfo(){
        cout<<"x="<<x<<" y="<<y<<" z="<<z;
    }
};
int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    Contoh aa;
    aa.tampilInfo();
    return a.exec();
}
```

Hasil:




Analisa

Terlihat bahwa kita bisa menginisialisasi isi dari variabel member yang kita miliki dengan cara menuliskannya pada bagian header method member seperti pada contoh diatas. Dan ketika class diinstasiasi maka otomatis konstruktor dipanggil dan semua nilai variabel member telah diinisialisasi seperti yang sudah dituliskan.

## const member method

Kita menggunakan kata kunci const untuk membuat suatu identifier konstanta. Konstanta berarti suatu variabel yang tidak bisa diganti / diubah nilainya pada saat program berjalan (runtime). Konstanta juga dapat digunakan pada method member Dengan memberikan kata kunci const setelah nama method, maka method tersebut juga tidak akan bisa diubah nilainya pada saat class dijalankan.

Kegunaan method const adalah pada asesor method. Mengapa? Karena pada asesor method kita menggunakan method tersebut untuk mengambil nilai dari member variabel, bukan untuk mengubah nilainya. Sedangkan pada mutator method, method tersebut tidak boleh dibuat const method karena method tersebut digunakan khusus untuk mengubah nilai dari member function.

Sehingga cara yang tepat untuk mendeklarasikan asesor method adalah dengan cara memberi kata kunci const pada akhir nama method tersebut. Contoh:

```
//mutator
void setUmur(int u);
//asesor
int getUmur() const;
```

### Labs 9. Penggunaan const method

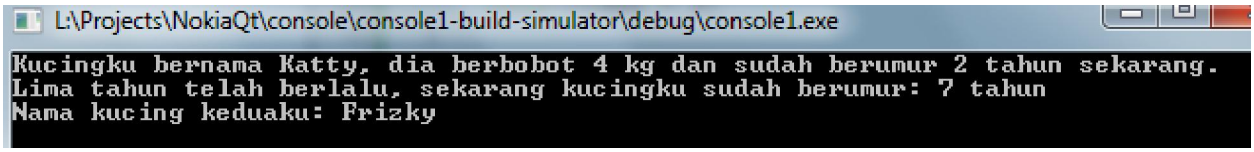
Buatlah program berikut ini:

```
#include <QtCore/QCoreApplication>
#include <iostream>
using namespace std;
class Kucing{
private:
    int umur;
```

```
    float berat;
    string nama;
public:
    //konstruktor
    Kucing(int umur);
    Kucing();
    //desktruktor
    ~Kucing();
    //asesor method
    int getUmur() const;
    float getBerat() const;
    string getNama() const;
    //mutator method
    void setUmur(int u);
    void setBerat(float b);
    void setNama(string s);
};
//implementasi konstruktor dan desktruktor
Kucing::Kucing(int u){
    this->umur = u;
}
Kucing::Kucing(){
}
Kucing::~Kucing(){
    cout<<"Obyek sudah dihancurkan!";
}
//implementasi function
int Kucing::getUmur() const{
    return this->umur;
}
float Kucing::getBerat() const{
    return this->berat;
}
string Kucing::getNama() const{
    return this->nama;
}
void Kucing::setUmur(int u){
    this->umur = u;
}
void Kucing::setBerat(float b){
    this->berat = b;
}
void Kucing::setNama(string s){
    this->nama = s;
}
int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    Kucing mycat(2);
    mycat.setNama("Katty");;
    mycat.setBerat(4);
    cout<<"Kucingku bernama "<<mycat.getNama()<<", dia berbobot "<<
            mycat.getBerat()<<" kg dan sudah berumur "<<mycat.getUmur()
            <<" tahun sekarang."<<endl;
```

```
    mycat.setUmur(7);
    cout<<"Lima tahun telah berlalu, sekarang kucingku sudah berumur:
"<<mycat.getUmur()<<" tahun"<<endl;
    Kucing kucingku2;
    kucingku2.setNama("Frizky");
    cout<<"Nama kucing keduaku: "<<kucingku2.getNama();
    return a.exec();
}
```

Hasil:




**Analisa:**

- Program diatas hasilnya sama dengan program sebelumnya karena kita hanya mengubah bagian asesor method dengan cara menambah kata const dibelakangnya. Bagian implementasi method tersebut juga harus disesuaikan.
- Dengan cara ini method asesor tersebut sudah bersifat read-only.

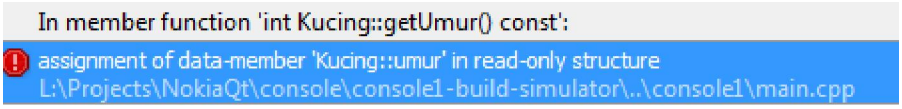
Ubahlah bagian method void getUmur() const;
Kita coba tambahkan baris program berikut sebelum return:

```
this->umur = 5.
```

Kode lengkapnya adalah:

```
int Kucing::getUmur() const{
    this->umur = 5;
    return this->umur;
}
```

Jika kita kompilasi program diatas, maka akan terjadi error sebagai berikut:




Mengapa hal ini terjadi? Karena method getUmur sudah dibuat menjadi konstan, yang artinya read-only. Di dalam method read-only kita tidak diperbolehkan melakukan operasi assigment atau pemberian nilai. Namun jika kita buang kata kunci const, maka method getUmur ini tetap dapat diubah nilainya. Dengan demikian kata kunci const benar-benar mampu mengamankan method dari hal yang tidak diinginkan, karena pada dasarnya method asesor memang tidak boleh mengubah nilai, hanya boleh membaca/mengambil nilai saja.

## Mendefinisikan Method Member

Selama ini kita mendefinisikan method member pada luar class. Selain cara diatas, kita juga bisa mendefinsikan method di dalam class itu sendiri secara langsung. Hal tersebut dinamakan inline

implementation. Contoh inline implementation adalah:

```
class Manusia{
private:
  string nama;
public:
void setNama(string n){
        this->nama = n;
  }
string getNama() const{
        return this->nama;
  }
}
```

Pada contoh diatas terlihat bahwa pada class Manusia implementasi kode method setNama dan getNama langsung dituliskan didalam program tersebut. Hal itu disebut inline implementation.

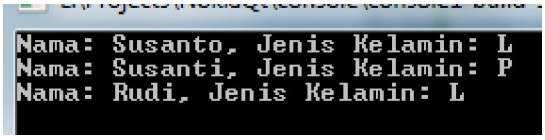
### Labs 10. Inline Implementation

Buatlah program berikut:

```
#include <QtCore/QCoreApplication>
#include <iostream>
using namespace std;
class Manusia{
private:
    string nama;
    char jenis_kelamin;
public:
    //konstruktor
    Manusia(){
    }
    Manusia(string nama){
        this->nama = nama;
    }
    //desktruktor
    ~Manusia(){
    }
    //accessor method
    string getNama() const{
        return this->nama;
    }
    char getJenis_Kelamin() const{
        return this->jenis_kelamin;
    }
    //mutator method
    void setNama(string n){
        this->nama = n;
    }
    void setJenis_Kelamin(char jk){
        this->jenis_kelamin = jk;
    }
    //method lain
```

```
    void tampilSemua(){
        cout<<"Nama: "<<this->getNama()<<", "<<
                "Jenis Kelamin: "<<this->getJenis_Kelamin()<<endl;
    }
};
int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    Manusia suami;
    suami.setNama("Susanto");
    suami.setJenis_Kelamin('L');
    Manusia istri("Susanti");
    istri.setJenis_Kelamin('P');
    Manusia anak("Rudi");
    anak.setJenis_Kelamin('L');
    suami.tampilSemua();
    istri.tampilSemua();
    anak.tampilSemua();
    return a.exec();
}
```

Hasil:




**Analisa:**

- Program diatas hanya menjelaskan bagaimana kita dapat mengimplementasikan method member langsung didalam tubuh class, tidak diluar class.
- Hal seperti ini biasa dilakukan pada bahasa pemrograman berorientasi obyek lain seperti misalnya Java.
- Inline implementation tidak berbeda dengan non-inline implementation.

## Class yang bertipe Class lain

Sangatlah mungkin kita membentuk class yang kompleks. Di dalam class tersebut member variabelnya dapat bertipe class lainnya. Contohnya adalah kita membuat class Mobil yang tentunya memiliki variabel member berupa class Roda, class Jok Mobil, class Mesin dan lain-lain. Contoh lain adalah class Garis yang terdiri dari class Titik. Class Bujursangkar juga dapat terdiri dari class Garis, dimana class Garis juga terdiri dari class Titik. Class dapat menjadi solusi yang baik untuk membuat tipe data baru yang memiliki member variabel dan member method yang tentunya sangat berguna.

### Labs 11. Class Mobil dan Class Roda

Buatlah program berikut:

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

```
class Roda{
private:
    string merk_roda;
    int diameter;
public:
    Roda(string merk,int diamtr){
        this->diameter = diamtr;
        this->merk_roda = merk;
    }
    Roda(){
    }
    string getMerk(){
        return this->merk_roda;
    }
    int getDiameter(){
        return this->diameter;
    }
    void setMerkRoda(string m){
        this->merk_roda = m;
    }
    void setDiameter(int d){
        this->diameter = d;
    }
};
class Mobil{
private:
    string merk_mobil;
    Roda roda_depan1;
    Roda roda_depan2;
    Roda roda_belakang1;
    Roda roda_belakang2;
public:
    //konstruktor
    Mobil(){
    }
    Mobil(string merk, Roda roda[4]){
        this->merk_mobil = merk;
        this->roda_depan1 = roda[0];
        this->roda_depan2 = roda[1];
        this->roda_belakang1 = roda[2];
        this->roda_belakang2 = roda[3];
    }
    Mobil(Roda r1, Roda r2, Roda r3, Roda r4){
        this->roda_depan1 = r1;
        this->roda_depan2 = r2;
        this->roda_belakang1 = r3;
        this->roda_belakang2 = r4;
    }
    //desktruktor
    ~Mobil(){
    }
    //accessor method
    string getMerkMobil() const{
        return this->merk_mobil;
```

```
    }
    //mutator method
    void setMerkMobil(string m){
        this->merk_mobil = m;
    }
    void setRoda(Roda rd[4]){
        this->roda_depan1 = rd[0];
        this->roda_depan2 = rd[1];
        this->roda_belakang1 = rd[2];
        this->roda_belakang2 = rd[3];
    }
    //method lain
    void tampilRoda(){
        cout<<"Roda depan1:"<<endl;
        cout<<"Merk: "<<this->roda_depan1.getMerk()<<endl;
        cout<<"Diameter: "<<this->roda_depan1.getDiameter()<<endl;
        cout<<"Roda depan2:"<<endl;
        cout<<"Merk: "<<this->roda_depan2.getMerk()<<endl;
        cout<<"Diameter: "<<this->roda_depan2.getDiameter()<<endl;
        cout<<"Roda belakang1:"<<endl;
        cout<<"Merk: "<<this->roda_belakang1.getMerk()<<endl;
        cout<<"Diameter: "<<this->roda_belakang1.getDiameter()<<endl;
        cout<<"Roda belakang2:"<<endl;
        cout<<"Merk: "<<this->roda_belakang2.getMerk()<<endl;
        cout<<"Diameter: "<<this->roda_belakang2.getDiameter()<<endl;
    }
    void tampilSemua(){
        cout<<"Merk: "<<this->getMerkMobil()<<endl;
        this->tampilRoda();
    }
};
int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    Roda r1("Bridgestone",40);
    Roda r2("Bridgestone",40);
    Roda r3("Bridgestone",40);
    Roda r4("Bridgestone",40);
    Mobil m1(r1,r2,r3,r4);
    m1.setMerkMobil("Innova");
    m1.tampilSemua();
    return a.exec();
}
```

Hasil:


**Analisa:**

- Program diatas mendemonstrasikan kepada kita bahwa kita dapat membuat class yang memiliki variabel member yang bertipe class lain.
- Cara mendeklarasikan variabel member bertipe class sama seperti cara mendefinisikan variabel member bertipe data biasa
- Variabel member yang bertipe data class akan memiliki sifat-sifat class tersebut.
- Pada contoh program diatas, class Mobil memiliki variabel member bertipe class Roda, maka variabel member roda_depan1, roda_depan2, roda_belakang1, dan roda_belakang2 akan memiliki sifat-sifat class Roda, dimana kita dapa mengakses semua variabel member class Roda dan juga method member class Roda.
- Cara mengakses variabel member dan method member class Roda sama seperti biasa, yaitu dengan menggunakan tanda titik (.). Namun perlu diingat bahwa kita tidak dapat langsung mengakses variabel member class Roda karena variabel member tersebut bersifat private. Yang dapat kita lakukan adalah mengakses method member yang menenkapsulasi variabel member class Roda. Pada contoh diatas kita mengakses method getMerk() dan getDiameter().

### Labs 12. Contoh Class Titik dan Garis

Buatlah program berikut ini:

```
#include <QtCore/QCoreApplication>
#include <iostream>
#include <math.h>
using namespace std;
class Titik{
private:
    int x;
    int y;
public:
    Titik(int x,int y){
        this->x = x;
        this->y = y;
    }
    Titik(){
    }
    int getX(){
        return this->x;
```

```
    }
    int getY(){
        return this->y;
    }
    void setX(int x){
        this->x=x;
    }
    void setY(int y){
        this->y=y;
    }
    //method tambahan
    void printPoint(){
        cout<<"("<<this->x<<" , "<<this->y<<")"<<endl;
    }
    bool isOrigin(){
        return (this->x == 0 && this->y == 0);
    }
};
class Garis{
private:
    Titik p1;
    Titik p2;
public:
    Garis(int x1,int y1,int x2,int y2) {
        p1.setX(x1);
        p1.setY(y1);
        p2.setX(x2);
        p2.setY(y2);
    }
    Garis(){
    }
    Garis(Titik t1,Titik t2){
        p1=t1;
        p2=t2;
    }
    void setPoint1(Titik p1){
        this->p1 = p1;
    }
    void setPoint2(Titik p2){
        this->p2 = p2;
    }
    void setPoints(Titik p1,Titik p2){
        this->p1 = p1;
        this->p2 = p2;
    }
    Titik getPoint1(){
        return this->p1;
    }
    Titik getPoint2(){
        return this->p2;
    }
    void printLine(){
        cout<<"awal: ";
        this->p1.printPoint();
```

```
        cout<<"akhir: ";
        this->p2.printPoint();
    }
    //Hitung panjang garis
    float getLength(){
        return sqrt((this->p1.getX() - this->p2.getX())*(this->p1.getX() - this-
>p2.getX()) + (this->p1.getY() - this->p2.getY())*(this->p1.getY() - this-
>p2.getY()));
    }
};
int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    Titik A(1,1);
    cout<<"A ";
    A.printPoint();
    Titik B(5,1);
    cout<<"B ";
    B.printPoint();
    Titik C(5,6);
    cout<<"C ";
    C.printPoint();
    Titik D(1,6);
    cout<<"D ";
    D.printPoint();
    Garis ab(A,B);
    cout<<"ab ";
    ab.printLine();
    cout<<"Panjang garis ab: "<<ab.getLength()<<endl;
    Garis bc(B,C);
    cout<<"bc ";
    bc.printLine();
    cout<<"Panjang garis bc: "<<bc.getLength()<<endl;
    Garis cd(D,C);
    cout<<"cd ";
    cd.printLine();
    cout<<"Panjang garis cd: "<<cd.getLength()<<endl;
    Garis da(D,A);
    cout<<"da ";
    da.printLine();
    cout<<"Panjang garis da: "<<da.getLength()<<endl;
    return a.exec();
}
```

Hasil:

```
A (1 , 1)
B (5 , 1)
C (5 , 6)
D (1 , 6)
ab awal: (1 , 1)
akhir: (5 , 1)
Panjang garis ab: 4
bc awal: (5 , 1)
akhir: (5 , 6)
Panjang garis bc: 5
cd awal: (1 , 6)
akhir: (5 , 6)
Panjang garis cd: 4
da awal: (1 , 6)
akhir: (1 , 1)
Panjang garis da: 5
```

**Analisa:**

- Program diatas juga menunjukkan contoh lain dari suatu class yang memiliki member variabel yang bertipe class lain. Pada contoh diatas class Garis memiliki variabel member yang berasal dari class Titik. Sehingga dari obyek Garis kita dapat mengakses semua method member class Titik.
- Dengan menggunakan rumus matematis perhitungan jarak antara dua buah koordinat (titik), maka kita bisa menghitung panjang garis. Untuk perhitungan dibutuhkan function sqrt yang berarti akar kuadrat, sehingga kita harus mengincludekan header math.h
- Method isOrigin pada class Titik digunakan untuk mengetahui apakah suatu koordinat berada di titik 0,0 atau tidak. Kita dapat menambahkan method lain yang sesuai kebutuhan kita.
- Di dalam kelas Garis kita memiliki beberapa konstruktor, ada yang tidak berparameter, ada yang berparameter 2 Titik dan berparameter 4 koordinat. Semuanya itu digunakan untuk tujuan yang sama, yaitu menciptakan obyek Titik pada member variabel class Garis, karena Garis pada dasarnya adalah terdiri dari 2 buah Titik.

TIPS

Pada bahasa C++ kita tidak dapat memanggil konstruktor dari dalam konstruktor lain yang berada dalam satu class.

Contoh:

```
class Halaman{
private:
    int nohal;
    int jenishal;
    //1 -> halaman biasa, 2 -> halaman header
public:
    Halaman(){
    }
    Halaman(int nohal){
        this->nohal = nohal;
    }
    Halaman(int nohal,int jenishal){
        Halaman(nohal);  //memanggil konstruktor Halaman diatasnya!
        this->jenishal = jenishal;
```

```
    }
};
```

Akan menghasilkan error!