



---

# Tutorial 2

---

Praktikum Pemrograman Berbasis Objek

Asisten IF2210 2023/2024

---



Gimana Praktikum 1 nya?

Banyak yang mendapatkan nilai sempurna nih,  
semoga konsisten yaa hingga praktikum terakhir :)

# Kesalahan Umum Praktikum 1

- Tidak teliti huruf besar/kecil pada output
- "krincing" dioutput hanya sekali, seharusnya dioutput n\_keys kali
- Tidak memahami perbedaan antara alokasi objek dengan new dan tanpa new
- Memanggil destruktur pada objek secara manual (misal p.~Paper()), bisa menyebabkan runtime error karena destruktur akan dipanggil kembali saat objek out of scope.
- Tidak tahu cara menggunakan private atribut static
- Kurang pahamnya konsep *friend*
- Kurang paham juga cara implementasi *overload*

# Inheritance

- Dalam paradigma pemrograman berorientasi objek, konsep Inheritance merupakan komponen penting
- Konsep ini memungkinkan kita untuk mendefinisikan suatu kelas berdasarkan sifat dan perilaku dari sebuah objek lain.
- Kelas yang menurunkan sifat disebut sebagai *parent class*, sedangkan kelas yang diturunkan disebut sebagai *child class*

## Contoh Inheritance: Kelas Plant

```
// plant.h
#ifndef PLANT_H
#define PLANT_H

class Plant {
protected:
    float waterStock;
public:
    Plant();
    void photosynthesis();
    void absorbWater(float water);
};

#endif
```

```
// plant.cpp
#include "plant.h"

Plant::Plant() {
    this->waterStock = 0;
}

void Plant::photosynthesis() {
    cout << "Nyam nyam makan cahaya" << endl;
}

void Plant::absorbWater(float water) {
    this->waterStock += water;
}
```

# Contoh Inheritance: Kelas Weed

```
// weed.h
#ifndef WEED_H
#define WEED_H
```

```
#include "plant.h"
```

```
class Weed : public Plant {
public:
    Weed();
    void photosynthesis();
    void step();
};
```

```
#endif
```

Turunan dari kelas Plant

Dapat mendefinisikan metode sendiri

```
// weed.cpp
#include "weed.h"
```

```
Weed::Weed() : Plant() {
    //
}
```

```
void Weed::photosynthesis() {
    Plant::photosynthesis();
    cout << "Smoke light everyday" << endl;
}
```

```
void Weed::step() {
    cout << "kresek..." << endl;
}
```

Memanggil metode *parent*

## Contoh Inheritance: Kelas FlyTrap

```
// flytrap.h
#ifndef FLYTRAP_H
#define FLYTRAP_H
```

```
#include "plant.h"
```

Definisi atribut sendiri

```
class FlyTrap : public Plant {
private:
    int eatenInsect;
public:
    FlyTrap();
    void eatInsect(int insect);
};
```

```
#endif
```

```
// flytrap.cpp
#include "flytrap.h"
```

```
FlyTrap::FlyTrap() : Plant() {
    this->eatenInsect = 0;
}
```

```
void FlyTrap::eatInsect(int insect) {
    this->eatenInsect += insect;
}
```

# Polymorphism

- Secara harfiah berarti banyak bentuk, diserap dari bahasa yunani
- Konsep ini menyatakan bahwa kelas anak bisa berlaku seperti kelas *parent*-nya



# Contoh Polymorphism

```
void waterPlant(Plant *v) {  
    v->absorbWater(1.5);  
    v->photosynthesis();  
}  
  
int main() {  
    Weed weed;  
    waterPlant(&weed);  
}
```

Output:

???

# Contoh Polymorphism

```
void waterPlant(Plant *v) {  
    v->absorbWater(1.5);  
    v->photosynthesis();  
}  
  
int main() {  
    Weed weed;  
    waterPlant(&weed);  
}
```

Output:

Nyam nyam makan cahaya

## Virtual Function

- Apabila metode kelas *parent* ditandai dengan *keyword* *virtual*, maka implementasi metode tersebut akan dilimpahkan ke kelas anak.
- Sehingga saat pemanggilan metode tersebut, definisi langsung dicari di kelas anak karena kelas anak bisa mendefinisikan implementasi metodenya dari kelas *parent*.

# Contoh Virtual Function

```
// plant.h
#ifndef PLANT_H
#define PLANT_H

class Plant {
protected:
    float waterStock;
public:
    Plant();
    virtual void photosynthesis();
    void absorbWater(float water);
};

#endif
```

Output:

Nyam nyam makan cahaya  
Smoke light everyday

## Contoh Virtual Function

```
void waterPlant(Plant *v) {  
    v->absorbWater(1.5);  
    v->photosynthesis();  
}  
  
int main() {  
    Weed weed;  
    waterPlant(&weed);  
}
```

Output:

Nyam nyam makan cahaya  
Smoke light everyday

# Perbandingan

```
void waterPlant(Plant v) {  
    v.absorbWater(1.5);  
    v.photosynthesis();  
}
```

```
void waterPlantPtr(Plant *v) {  
    v->absorbWater(1.5);  
    v->photosynthesis();  
}
```

```
void waterPlantRfc(Plant &v) {  
    v.absorbWater(1.5);  
    v.photosynthesis();  
}
```

```
int main() {  
    Weed weed;  
    waterPlant(weed);  
    cout << "----" << endl;  
    waterPlantPtr(&weed);  
    cout << "----" << endl;  
    waterPlantRfc(weed);  
  
    return 0;  
}
```

Output???

photosynthesis() virtual

# Perbandingan

```
int main() {  
    Weed weed;  
    waterPlant(weed);  
    cout << "----" << endl;  
    waterPlantPtr(&weed);  
    cout << "----" << endl;  
    waterPlantRfc(weed);  
  
    return 0;  
}
```

Nyam nyam makan cahaya

---

Nyam nyam makan cahaya

Smoke light everyday

---

Nyam nyam makan cahaya

Smoke light everyday

Ada yang tau kenapa begini?

## Contoh Polymorphism, Inheritance & Virtual

```
void waterPlant(Plant v) {  
    v.absorbWater(1.5);  
    v.photosynthesis();  
}  
  
int main() {  
    Plant p;  
    p.photosynthesis();  
    FlyTrap ft;  
    waterPlant(ft);  
    ft.eatInsect(5);  
    Weed w;  
    waterPlant(w);  
    Plant p2 = w;  
    p2.photosynthesis();  
}
```

Nyam nyam makan cahaya  
Nyam nyam makan cahaya  
Nyam nyam makan cahaya  
Nyam nyam makan cahaya



## Virtual Destructor

- Bagaimana jika destructor? Apakah perlu diberi virtual?
  - Jika tidak diberi virtual, maka destructor yang dipanggil hanya destructor kelas parent.
  - Dengan demikian, beberapa hal yang perlu di-*free* oleh kelas anak, tidak dilaksanakan.
  - Secara praktik, menjadikan destructor virtual pada kelas parent lebih baik.

## Pure Virtual Function

- Pure Virtual function merupakan metode yang harus didefinisikan di kelas anak. Kelas *parent* cukup mendefinisikan nama metode saja, dan implementasinya wajib dilakukan di kelas anak.
- Misal metode **getSpecies** harus diimplementasikan di kelas anak, tidak masuk akal apabila diimplementasikan di kelas Plant
- Kelas yang memiliki setidaknya satu pure virtual function disebut sebagai **abstract class**, kelas yang tidak dapat diinstansiasikan
- Di bahasa lain, ini juga sering disebut sebagai **abstract class**

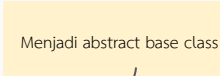
# Contoh Pure Virtual Function

```
// plant.h
#ifndef PLANT_H
#define PLANT_H

class Plant {
protected:
    float waterStock;
public:
    Plant();
    virtual void photosynthesis();
    void absorbWater(float water);
    virtual char* getSpecies() = 0;
};

#endif
```

Menjadi abstract base class



```
// weed.cpp
char* Weed::getSpecies() {
    return "Cannabis sativa";
}

// flytrap.cpp
char* FlyTrap::getSpecies() {
    return "Dionaea muscipula";
}
```

## Contoh Polymorphism, Inheritance & Virtual 1

```
void waterPlant(Plant *v) {  
    v->absorbWater(1.5);  
    v->photosynthesis();  
}  
  
int main() {  
    Plant p;  
    p.photosynthesis();  
    FlyTrap ft;  
    waterPlant(&ft);  
    ft.eatInsect(5);  
    Weed w;  
    waterPlant(&w);  
    Plant p2 = w;  
    p2.photosynthesis();  
}
```

Kalau photosynthesis() jadi tidak virtual, outputnya apa?

# Contoh Polymorphism, Inheritance & Virtual 1

```
void waterPlant(Plant *v) {  
    v->absorbWater(1.5);  
    v->photosynthesis();  
}  
  
int main() {  
    Plant p;  
    p.photosynthesis();  
    FlyTrap ft;  
    waterPlant(&ft);  
    ft.eatInsect(5);  
    Weed w;  
    waterPlant(&w);  
    Plant p2 = w;  
    p2.photosynthesis();  
}
```

Nyam nyam makan cahaya  
Nyam nyam makan cahaya  
Nyam nyam makan cahaya  
Nyam nyam makan cahaya

## Contoh Polymorphism, Inheritance & Virtual 2

```
void waterPlant(Plant *v) {  
    v->absorbWater(1.5);  
    v->photosynthesis();  
}  
  
int main() {  
    Plant p;  
    p.photosynthesis();  
    FlyTrap ft;  
    waterPlant(&ft);  
    ft.eatInsect(5);  
    Weed w;  
    waterPlant(&w);  
    Plant p2 = w;  
    p2.photosynthesis();  
}
```

Kalau photosynthesis() jadi virtual, outputnya apa?

## Contoh Polymorphism, Inheritance & Virtual 2

```
void waterPlant(Plant *v) {  
    v->absorbWater(1.5);  
    v->photosynthesis();  
}  
  
int main() {  
    Plant p;  
    p.photosynthesis();  
    FlyTrap ft;  
    waterPlant(&ft);  
    ft.eatInsect(5);  
    Weed w;  
    waterPlant(&w);  
    Plant p2 = w;  
    p2.photosynthesis();  
}
```

Nyam nyam makan cahaya  
Nyam nyam makan cahaya  
Nyam nyam makan cahaya  
Smoke light everyday  
Nyam nyam makan cahaya

## Contoh Pure Virtual

```
void waterPlant(Plant *v) {  
    v->absorbWater(1.5);  
    v->photosynthesis();  
}  
  
int main() {  
    Plant p;  
    p.photosynthesis();  
    FlyTrap ft;  
    waterPlant(&ft);  
    ft.eatInsect(5);  
    Weed w;  
    waterPlant(&w);  
    Plant p2 = w;  
    p2.photosynthesis();  
}
```

Jangan lupa, diganti di:

```
void Weed::photosynthesis() {  
    Plant::photosynthesis();  
    cout << "Smoke light everyday" << endl;  
}
```

Jangan manggil superclass, karna superclassnya virtual

Kalau photosynthesis() jadi pure virtual, outputnya apa?



## Contoh Pure Virtual 0

```
void waterPlant(Plant *v) {  
    v->absorbWater(1.5);  
    v->photosynthesis();  
}  
  
int main() {  
    Plant p;  
    p.photosynthesis();  
    FlyTrap ft;  
    waterPlant(&ft);  
    ft.eatInsect(5);  
    Weed w;  
    waterPlant(&w);  
    Plant p2 = w;  
    p2.photosynthesis();  
}
```

Jangan lupa, diganti di:

```
void Weed::photosynthesis() {  
    Plant::photosynthesis();  
    cout << "Smoke light everyday" << endl;  
}
```

Jangan manggil superclass, karna superclassnya virtual

Error: cannot declare variable  
'p' to be of abstract type 'Plant'

## Contoh Pure Virtual 1

```
void waterPlant(Plant *v) {  
    v->absorbWater(1.5);  
    v->photosynthesis();  
}  
  
int main() {  
    Weed w;  
    waterPlant(&w);  
    Plant *p2 = &w;  
    p2->photosynthesis();  
    Plant *p3;  
    p3 = &w;  
    p3->photosynthesis();  
}
```

Jangan lupa, diganti di:

```
void Weed::photosynthesis() {  
    Plant::photosynthesis();  
    cout << "Smoke light everyday" << endl;  
}
```

Jangan memanggil *method* punya *superclass*, karna *superclass photosynthesis* adalah *pure virtual*

Kalau begini, bisa gak? Masih error?

photosynthesis() pure virtual.

Jadi apa outputnya???

## Contoh Pure Virtual 1

```
void waterPlant(Plant *v) {  
    v->absorbWater(1.5);  
    v->photosynthesis();  
}  
  
int main() {  
    Weed w;  
    waterPlant(&w);  
    Plant *p2 = &w;  
    p2->photosynthesis();  
    Plant *p3;  
    p3 = &w;  
    p3->photosynthesis();  
}
```

Jangan lupa, diganti di:

```
void Weed::photosynthesis() {  
    Plant::photosynthesis();  
    cout << "Smoke light everyday" << endl;  
}
```

Jangan memanggil method punya superclass, karna superclass photosynthesis adalah pure virtual

Smoke light everyday  
Smoke light everyday  
Smoke light everyday

# Pure Virtual & Virtual

Ada ide gunanya virtual & pure virtual ini apa selain destructor?

Abstraction

Misal ada abstract class Shape, punya method draw()

Parent class-nya gak bisa definisi draw apa karena abstrak, ada yang bisa gambar Shape gimana?

Tapi saat diturunkan menjadi Circle, Rectangle, Triangle dsb. Bisa didefinisikan drawnya seperti apa. Dan setiap child class yang merupakan Shape pasti bisa digambar (tidak menyalahi aturan Inheritance).

# Presensi

Waktunya presensi!

---



# Sekian.

Ditunggu praktikum dan tutorial berikutnya.

---