



IPB University
— Bogor Indonesia —

KOM120C -- BAHASA PEMROGRAMAN

Paradigma Pemrograman

- Pengantar Pemrograman
 - Paradigma Pemrograman
 - Pemrograman C++
-

Tim Pengajar Bahasa Pemrograman IPB University



BAHASA PEMROGRAMAN

- Bahasa pemrograman adalah suatu **sistem notasi** untuk menuliskan tugas komputasi yang harus dilaksanakan oleh mesin, dan bentuknya dapat dibaca serta dipahami oleh manusia.
- Beberapa bahasa komputer dirancang untuk memfasilitasi operasi-operasi tertentu, misalnya komputasi numerik, manipulasi teks, I/O, etc.
- Pada umumnya, bahasa pemrograman komputer biasanya dirancang dengan menggunakan **paradigma pemrograman** tertentu. Artinya mengikuti aliran atau **genre** tertentu.

PRINSIP PERANCANGAN BAHASA PEMROGRAMAN

Prinsip perancangan bahasa pemrograman:

(1) **Sintaks**, (2) **Nama dan Tipe**, (3) **Semantik**.

- **Sintaks** menjelaskan bagaimana struktur program yang benar.
Struktur bahasa pemrograman modern didefinisikan menggunakan bahasa formal yang disebut context-free-grammar.
- **Nama dan Tipe** menunjukkan bagaimana aturan penamaan entitas (variabel, fungsi, class, parameter, dsb).
- **Semantik**, arti dari program. Ketika program dijalankan, efek tiap instruksi didefinisikan oleh semantik dari bahasa.

DESAIN BAHASA PEMROGRAMAN

Desain bahasa pemrograman memperhatikan 3 hal:

- **Architecture.** Bahasa pemrograman dirancang untuk komputer: *well-match* atau tidak dengan arsitektur komputer yang ada?
- **Technical Setting,** memperhatikan sistem operasi, IDE (Integrated Development Environment), network, dan referensi lingkungan lainnya.
- **Standards:** ANSI (American National Standards Institute), atau ISO (International Standards Organization). Contoh: ISO Pascal (1990), ANSI/ISO C++ (2003), dsb.

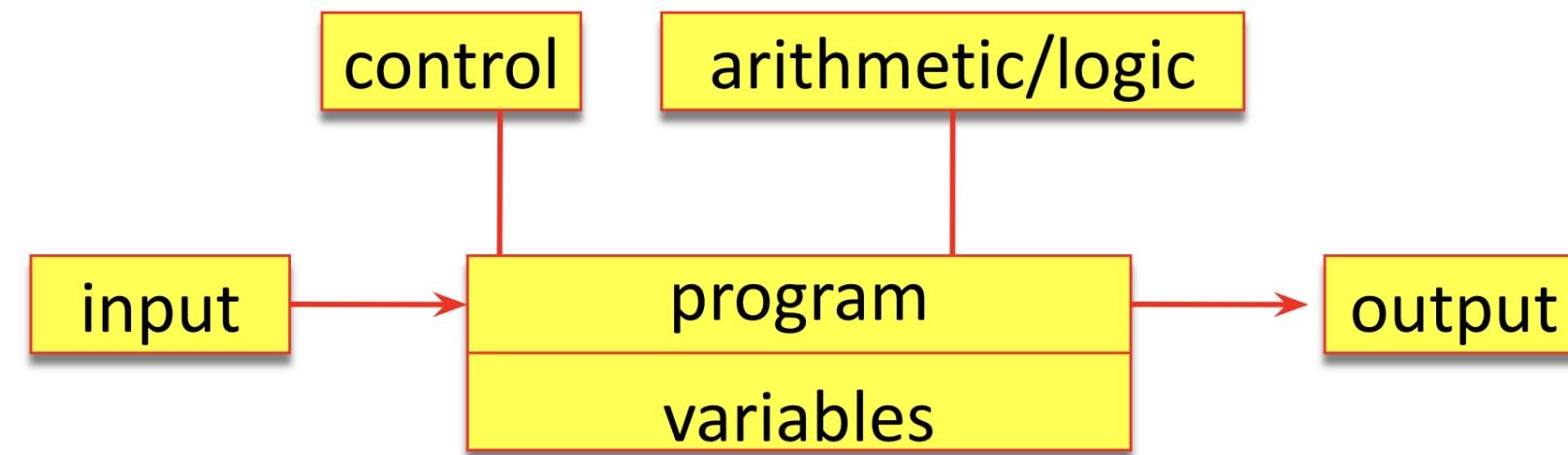
PARADIGMA BAHASA PEMROGRAMAN

Paradigma pemrograman adalah bentuk pemecahan masalah mengikuti aliran atau **genre** tertentu dari program dan bahasa.

Imperative/ Algorithmic	Declarative		Object-Oriented
	Functional Programming	Logic Programming	
Algol Cobol PL/1 Ada C Pascal Modula-3	Lisp Haskell ML Miranda APL	Prolog	SmallTalk Simula C++ Java

Imperative/Algorithmic

- Paradigma paling tua, didasari oleh model komputasi klasik von Neumann-Eckert: INPUT → PROSES → OUTPUT.
- Program dan variabel disimpan bersama.
- Program terdiri dari instruksi yang membentuk perhitungan, *assignment*, input, output, dan kontrol.



Serba Fungsi

Mulai dikembangkan tahun 1960an, dimotivasi oleh peneliti bidang artificial intelligence, symbolic computation, theorem proving, rule-based system, dan NLP.

LISP

List programming

Functional Programming

Struktur

Memodelkan masalah komputasi sebagai suatu **fungsi matematika**, yang mempunyai input (domain) dan hasil atau output (range).

Buat program menentukan bilangan terkecil dari tiga bilangan.

```
> (min 6 4 6)  
4
```

```
#lang scheme  
(define (min a b c)  
  ( if (and (< a b) (< a c))  
        a  
        ( if (< b c) b c )  
  ))
```

Logic Programming

Buat program menghitung banyaknya elemen list yang nilainya kurang dari suatu nilai tertentu

```
?- hitung(10, [1,2,15,7,25,10], X).  
X=3
```

```
hitung([_|T], X) :-  
    H<A, !, hitung(A,T,Y), X is Y+1.  
hitung([_|T], X) :- hitung(T, X).
```

- **Pemrograman deklaratif**, mendeklarasikan tujuan komputasi, bukan menyusun algoritme secara detil. Disebut juga **rule-based programming**.
- LP dirancang untuk mendeskripsikan properti dari suatu obyek. Hubungan antar obyek dinyatakan dengan aturan if-then (jika-maka).

Pemrograman Berorientasi Objek (PBO)

Metode pemrograman yang berorientasikan kepada **objek**, dimana semua **data** dan **fungsi** dalam metode ini didefinisikan ke dalam kelas-kelas atau objek-objek agar bisa saling bekerjasama dalam memecahkan masalah.

Bandingkan dengan pemrograman terstruktur (*algorithmic*) seperti yang dilakukan sebelumnya dalam KOM120B !

Dibutuhkan pemahaman yang dalam tentang objek dalam pemecahan persoalan → termasuk pemrograman tingkat tinggi



Pengantar ke OOP **Pemrograman C++**

Baca di sini:
<https://cplusplus.com/doc/turorial>

C++ as a Better C

Program sederhana C++

```
#include < iostream >
int main()
{
    Namespace int a;
    std::cin >> a;
    int b;
    std::cin >> b;
    int sum=a+b;
    std::cout << sum << std::endl;
    return 0;
}
```

Annotations:

- Namespace: Points to the word "Namespace" in the code.
- Standard input/output stream library: Points to the line "#include < iostream >".
- Standard input stream: Points to the line "std::cin >> a;".
- Standard output stream: Points to the line "std::cout << sum << std::endl;".
- Inserts a new-line character and flushes the stream: Points to the line "std::endl;".

Bandingkan dengan C !

Data Type Scope

Bebas mendeklarasikan variable.

Perhatikan ruang lingkup dari variable yang didefinisikan (local, global, segment).

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

int main() {
    // ...
    for (int i=0; i<5; i++)
    {
        // ...
    }
    return 0;
}
```

Variabel i hanya dikenal di area ini

Fundamental Data Types

Group	Type names*	Notes on size / precision
Character types	<code>char</code>	Exactly one byte in size. At least 8 bits.
	<code>char16_t</code>	Not smaller than <code>char</code> . At least 16 bits.
	<code>char32_t</code>	Not smaller than <code>char16_t</code> . At least 32 bits.
	<code>wchar_t</code>	Can represent the largest supported character set.
Integer types (signed)	<code>signed char</code>	Same size as <code>char</code> . At least 8 bits.
	<code>signed short int</code>	Not smaller than <code>char</code> . At least 16 bits.
	<code>signed int</code>	Not smaller than <code>short</code> . At least 16 bits.
	<code>signed long int</code>	Not smaller than <code>int</code> . At least 32 bits.
	<code>signed long long int</code>	Not smaller than <code>long</code> . At least 64 bits.
Integer types (unsigned)	<code>unsigned char</code>	(same size as their signed counterparts)
	<code>unsigned short int</code>	
	<code>unsigned int</code>	
	<code>unsigned long int</code>	
	<code>unsigned long long int</code>	
Floating-point types	<code>float</code>	
	<code>double</code>	Precision not less than <code>float</code>
	<code>long double</code>	Precision not less than <code>double</code>
Boolean type	<code>bool</code>	
Void type	<code>void</code>	no storage
Null pointer	<code>decltype(nullptr)</code>	

Function Overloading

Perbedaan fungsi dilihat dari prototype-nya (return type, function name, argument).
Memungkinkan membuat fungsi dengan nama yang sama.

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

int kuadrat(int x) { return x*x; }
double kuadrat(double x) { return x*x; }

int main() {
    cout << kuadrat(7) << endl
        << kuadrat(7.5) << endl;
    return 0;
}
```

Function Template

To pass data type as a parameter.

Memungkinkan membuat sebuah fungsi yang berlaku untuk beberapa tipe data berbeda.

```
#include <iostream>
using namespace std;
template <typename T>
T maksimum(T n1, T n2, T n3) {
    T besar = n1;
    if (n2>besar) besar=n2;
    if (n3>besar) besar=n3;
    return besar;
}
int main() {
    int i1, i2, i3;
    cin >> i1 >> i2 >> i3;
    cout << maksimum(i1,i2,i3) << endl << endl;
    double d1, d2, d3;
    cin >> d1 >> d2 >> d3;
    cout << maksimum(d1,d2,d3) << endl;
    return 0;
}
```

Latihan 1: Gunakan C++

Buat program membaca n bilangan bulat dan menghitung rata-rata dari bilangan-bilangan yang berada di posisi kelipatan k. Jika n = 0 atau tidak ada data pada posisi tersebut, maka rata-rata bernilai 0.
Batasan: $0 < n < 2M$, $1 < k < (n-1)$

Contoh Input:

7 2
10 20 30 40 50 60 70

Contoh Output:

40.00

Latihan 2 : Gunakan C++

Diketahui 2 deretan bilangan yang sudah terurut ascending, masing-masing diakhiri dengan -9. Buat program mencetak seluruh bilangan yang ada secara terurut ascending juga.
[TIDAK BOLEH ADA PROSES SORTING ARRAY]

Contoh Input:

2 5 9 -9

4 8 10 15 20 -9

Contoh Output:

2 4 5 8 9 10 15 20



IPB University
— Bogor Indonesia —

KOM120C -- BAHASA PEMROGRAMAN

Pemrograman Berorientasi Objek

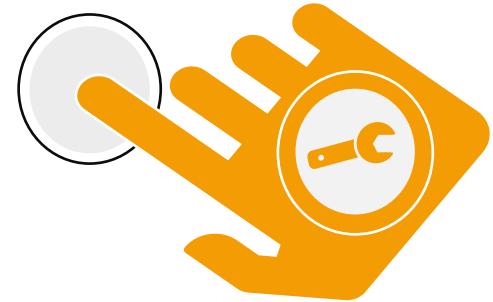
- Prinsip Dasar OOP
- Object and Class
- Struct versus Class

Tim Pengajar Bahasa Pemrograman IPB University

PRINSIP DASAR OOP

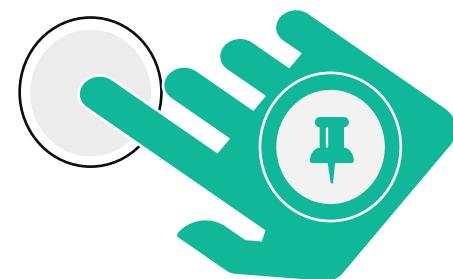
POLYMORPHISM

Tiap objek tahu siapa dirinya



ENCAPSULATION

- Membungkus prosedur dan data dalam satu objek.
- OOP memodelkan objek yang ada di dunia nyata ke dalam software objek dalam pemrograman.
- Implementasi dalam bentuk **Class**.
- Berfungsi sebagai ADT (*Abstract Data Type*)



INHERITANCE

- Pewarisan sifat objek
- Mengembangkan class baru dari class yang sudah ada.

Contoh Problem

Menjumlahkan dua bilangan



Paradigma Prosedural

Berfikir *algorithmic* → *imperative programming* (prosedural):

- **Input**. Apa inputnya? : dua bilangan bulat, mis. a dan b
- **Output**. Apa outputnya? : sebuah bilangan hasil, mis. sum
- **Proses**. Bagaimana langkah-langkah perhitungannya? :
 $\text{sum} = \text{a} + \text{b}$

Solusi → dalam bentuk algoritme:

```
read(a,b)  
sum=a+b  
write(sum)
```

Kode program:

```
#include <iostream>  
int main() {  
    int a,b;  
    std::cin >> a >> b;  
    int sum=a+b;  
    std::cout << sum << '\n';  
    return 0;  
}
```

Contoh Problem

Menjumlahkan dua bilangan



Saya harus membuat “benda” (objek) yang:

- memiliki dua variabel bilangan bulat
- dapat dimasukkan (menerima) dua nilai
- dapat dimasukkan (menerima) nilai yang pertama
- dapat dimasukkan (menerima) nilai yang kedua
- dapat memberikan hasil penjumlahan dua nilai



data / atribut

prosedur / behaviour / methods

Enkapsulasi adalah membungkus **data/atribut** dan **prosedur** ke dalam sebuah **objek**

Membuat Objek dalam Pemrograman

Objek diimplementasikan dalam bentuk **class**.

Struktur class dalam C++:

```
class <class-name> {  
    // data/atribut  
    ...  
    ...  
    // prosedur/fungsi  
    ...  
    ...  
}
```

Setiap anggota atau elemen dari class jenis akses (*access modifier*), yang menunjukkan bisa tidaknya elemen tersebut diakses suatu class:



private

hanya dapat diakses di dalam class itu sendiri



protected

dapat diakses oleh class itu sendiri beserta turunannya



public

dapat diakses di luar class

By default, akses terhadap elemen class adalah **private**.

Class myClass

Contoh problem menjumlahkan dua bilangan

Saya harus membuat "benda" (objek) yang:

- memiliki dua variabel bilangan bulat
- dapat dimasukkan (menerima) dua nilai
- dapat dimasukkan (menerima) nilai yang pertama
- dapat dimasukkan (menerima) nilai yang kedua
- dapat memberikan hasil penjumlahan dua nilai

```
class myClass {  
    private:  
        int a,b;  
    public:  
        void set(int p1, int p2)  
        { a=p1; b=p2; }  
        void setA(int p) { a=p; }  
        void setB(int p) { b=p; }  
        int sum() { return a+b; }  
}
```

Menguji Class

Contoh program
menjumlahkan dua bilangan

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

class myClass
{
    private:
        int a,b;
    public:
        void set(int p1, int p2) { a=p1; b=p2; }
        void setA(int p) { a=p; }
        void setB(int p) { b=p; }
        int sum() { return a+b; }
};

// driver
int main()
{
    myClass myObj; // myClass sebagai ADT
    myObj.set(5,10);
    cout << myObj.sum() << endl;
    myObj.setB(3);
    cout << myObj.sum() << endl;
    return 0;
}
```

Create object-instance → instansiasi

Memberikan nilai 5 dan 10 ke dalam objek

Mengganti nilai kedua dari objek myObj

Constructor

```
int main()
{
    myClass myObj;
    myObj.set(5,10);
    // ...
    return 0;
}
```

Pada saat dibuat objek myObj, berapa nilai data a dan b ?

Perlukah dibuat inisialisasi nilai
a dan b pada saat dibuat objek myObj
pertama kali?

Constructor

Apakah bisa dibuat objek myObj
sekaligus memberi nilai atributnya ?

Constructor

Constructor adalah fungsi yang otomatis diakses pada saat instansiasi.
Nama fungsi constructor **sama dengan nama class**.
Sering menggunakan konsep ***function overloading***.

Class myClass

Memiliki constructor

```
class myClass
{
    private:
        int a,b;
    public:
        myClass() { a=b=0; } // default constructor
        myClass(int p1, int p2) { a=p1; b=p2; }
        void set(int p1, int p2) { a=p1; b=p2; }
        void setA(int p) { a=p; }
        void setB(int p) { b=p; }
        int sum() { return a+b; }
};

int main()
{
    myClass myObj1;      // membuat objek myObj
    myClass myObj2(5,10);
    cout << myObj1.sum() << endl;
    cout << myObj2.sum() << endl;
    return 0;
}
```

Membuat Objek

Automatic and Dynamic (Pointer)

```
// Create automatic storage object  
myClass obj1;  
obj1.set(5,10);  
  
// Create dynamic storage object  
myClass* obj2=new myClass();  
obj2->set(5,10);  
  
// Use constructor  
myClass obj3(5,10);  
myClass* obj4=new myClass(5,10);
```

STRUCT versus CLASS

Struct hanya berisi data/atribut

```
struct Bilangan  
{  
    int a;  
    int b;  
};
```

Class berisi data/atribut dan/atau **prosedur**

```
class Bilangan  
{  
    int a;  
    int b;  
public:  
    Bilangan() { a=b=0; }  
    // dst.  
};
```

Setiap elemen struct adalah **public**.

Setiap elemen class bisa **private, protected, public**.

Struct tidak dapat memiliki turunan
(**inheritance**).

Class dapat memiliki turunan (**inheritance**)

Struct dan Class, keduanya berfungsi sebagai ADT (Abstract Data Type).



STRUCT versus CLASS

Struct dapat diimplementasikan menggunakan Class. Itu kenapa di Java tidak ada Struct.

```
struct Bilangan
{
    int a;
    int b;
};

int main()
{
    struct Bilangan obj;
    obj.a=5;
    obj.b=10;
    // ...
    // ...
    return 0;
}
```

```
class Bilangan
{
    int a;
    int b;
};

int main()
{
    Bilangan obj;
    obj.a=5;
    obj.b=10;
    // ...
    // ...
    return 0;
}
```

ERROR.
Mengapa?

STRUCT versus CLASS

Struct dapat diimplementasikan menggunakan Class. Itu kenapa di Java tidak ada Struct.

```
struct Bilangan
{
    int a;
    int b;
};

int main()
{
    struct Bilangan obj;
    obj.a=5;
    obj.b=10;
    // ...
    // ...
    return 0;
}
```

```
class Bilangan {
public:
    int a;
    int b;
};

int main()
{
    Bilangan obj;
    obj.a=5;
    obj.b=10;
    // ...
    // ...
    return 0;
}
```

Class sudah
identic dengan
Struct

BENAR

Tahapan OOP

Tahapan yang umum dilakukan untuk menyelesaikan persoalan menggunakan OOP



Object Design

Merancang objek (*attribute and behaviour*)

Menyusun hubungan antar objek (inheritance, polymorphism)



Create Class

Implementasi objek dalam pemrograman



Driver

Menyusun instruksi untuk memecahkan persoalan dengan menggunakan class yang sudah dibuat. Disini tempat untuk membuat instance (object).



Testing

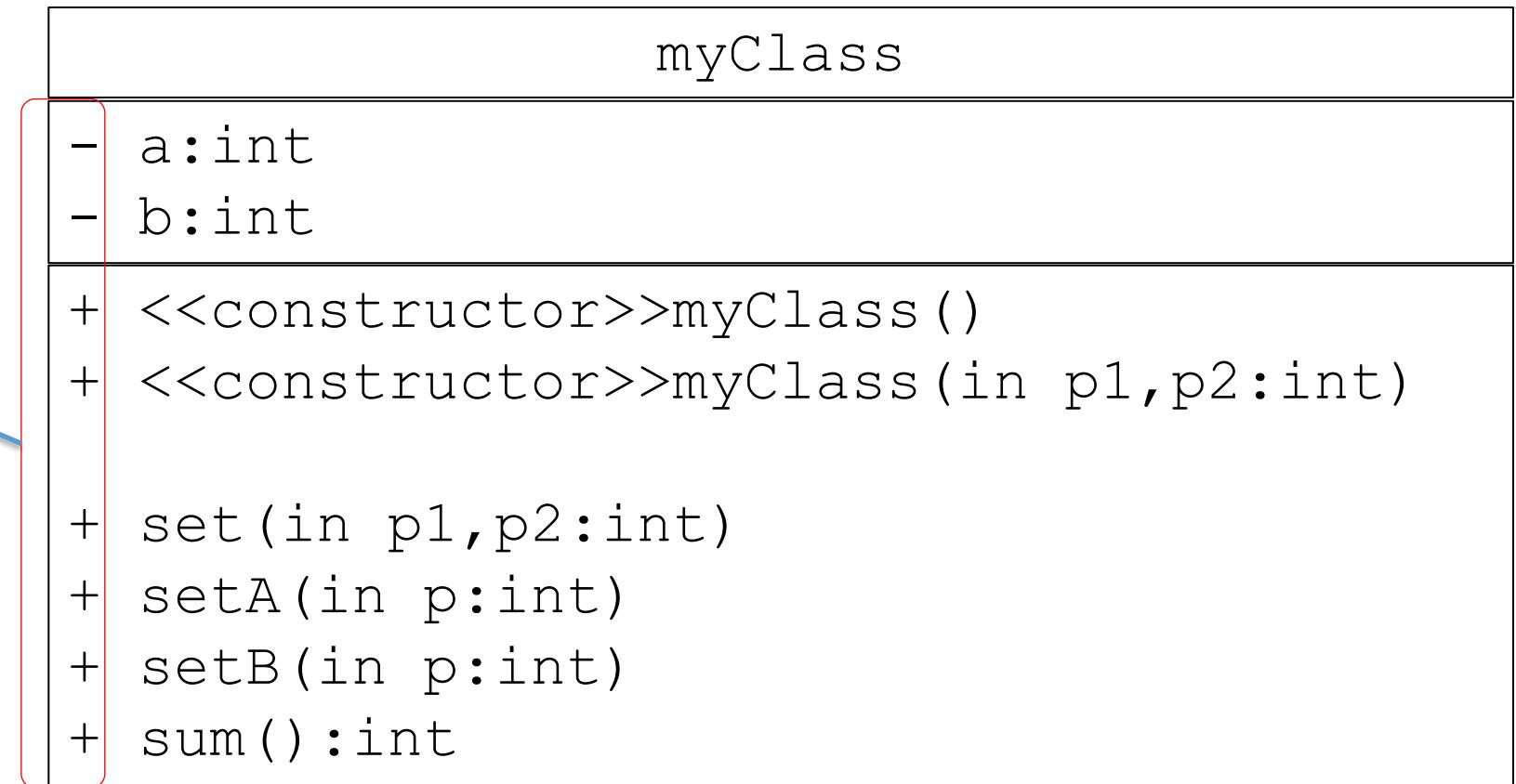
Menguji program dengan menggunakan kasus data.

Object Design

Merancang objek, salah satunya menggunakan presentasi UML (*Unified Modeling Language*)

Access modifier

- + : public
- : private
- # : protected



Latihan → Diskusi Kelas

Rancanglah suatu pendekatan OOP untuk menyelesaikan persoalan berikut:

Buat program untuk membuat dan mengolah nilai sebuah counter. Nilai counter dapat dinaikkan satu satuan, diturunkan satu satuan, dan ditampilkan nilainya. Nilai awal counter dapat dibuat dengan nilai tertentu.

Pengolahan terhadap counter tersebut menggunakan kode operasi seperti berikut:

- 0 n membuat counter baru dengan nilai awal n
- 1 menaikkan counter satu satuan
- -1 menurunkan counter satu satuan
- 9 menampilkan nilai counter
- -9 akhir dari input data

Contoh input:

0 5
1
1
9
-1
9
0 0
-1
1
9
-9

Contoh output:

7
6
1



IPB University
— Bogor Indonesia —

KOM120C -- BAHASA PEMROGRAMAN

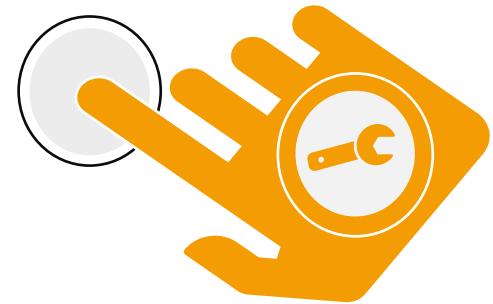
Object and Class

- Review OOP
 - Object and Class
-

Review : PRINSIP DASAR OOP

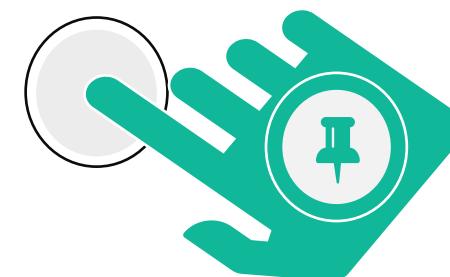
POLYMORPHISM

Tiap objek tahu siapa dirinya



ENCAPSULATION

- Membungkus prosedur dan data dalam satu objek.
- OOP memodelkan objek yang ada di dunia nyata ke dalam software objek dalam pemrograman.
- Implementasi dalam bentuk **Class**.
- Berfungsi sebagai ADT (*Abstract Data Type*)



INHERITANCE

- Pewarisan sifat objek
- Mengembangkan class baru dari class yang sudah ada.

Review : Membuat Objek dalam Pemrograman

Objek diimplementasikan dalam bentuk **class**.

Struktur class dalam C++:

```
class <class-name> {  
    // data/atribut  
    ...  
    ...  
    // prosedur/fungsi  
    ...  
    ...  
}
```

Setiap anggota atau elemen dari class jenis akses (*access modifier*), yang menunjukkan bisa tidaknya elemen tersebut diakses suatu class:



private

hanya dapat diakses di dalam class itu sendiri



protected

dapat diakses oleh class itu sendiri beserta turunannya



public

dapat diakses di luar class

By default, akses terhadap elemen class adalah **private**.

Contoh Kasus: Objek "person"

Materi Praktikum 02 : Class Person

Deskripsi

Buatlah program C++ yang mengimplementasikan sebuah class bernama Person, yang memiliki atribut nama (string), usia (int), tinggi (int), dan berat (double). Default constructor class ini akan secara otomatis mengisi atribut nama dengan string kosong, dan atribut lainnya diisi dengan nilai 0 (nol). Class Person memiliki prosedur bernama setPerson untuk mengisi nilai setiap atribut. Tambahkan prosedur lainnya agar dapat memproses apa yang diminta oleh program utama (main) yang diuraikan di bawah ini.

Program akan membaca n baris data yang berisi nama, usia, tinggi badan, dan berat badan. Gunakan class Person untuk menyimpan n data tersebut. Selanjutnya program akan menuliskan seluruh nama dan usia, dan dilanjutkan dengan menuliskan rata-rata tinggi badan, ~~jumlah~~ banyaknya orang (person) yang tinggi badannya di atas nilai rata-rata. Program harus benar-benar mengimplementasikan class bagi objek Person. Oleh karena itu, tidak diperkenankan mengolah data yang berada dari variabel biasa (bukan atribut dari class). Semua atribut class dikelompokkan sebagai private.

Format Masukan

Baris pertama adalah sebuah bilangan bulat n, $1 \leq n \leq 100$, yang menunjukkan banyaknya baris data yang akan dibaca. Dan n baris berikutnya berisi data nama, usia, tinggi badan, dan berat badan yang masing-masing dipisahkan oleh satu spasi. Nama orang berupa string dengan hanya terdiri atas satu kata.

Sebagai implementasi software dari objek "person" atau "orang".

Memiliki atribut : **nama, usia, tinggi, berat**.

Memiliki behaviour (perilaku, methods, etc):

- Inisialisasi (constructor)
- Dapat diberi nilai setiap atribut (mutator) : `setPerson()`
- Dapat diakses nilai setiap atribut (accessor)

Butuh sebuah class lain sebagai implementasi dari objek "beberapa person "

Desain Class Person

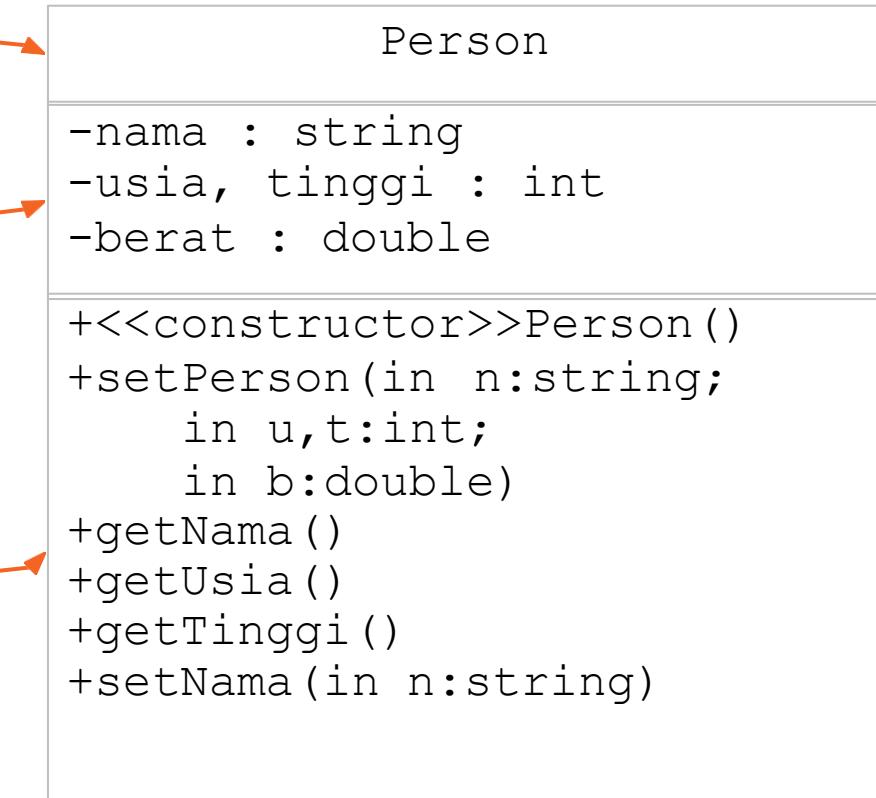
Materi Praktikum 02 : Class Person

Sebagai implementasi software dari objek "person" atau "orang".

Memiliki atribut : **nama, usia, tinggi, berat**.

Memiliki behaviour (perilaku, methods, etc) :

- Inisialisasi (constructor)
- Dapat diberi nilai atribut (mutator)
- Dapat diakses nilai atribut (accessor)



Presentasi berupa UML
(*Unified Modeling Language*)

Class Person

Materi Praktikum 02 : Class Person

```
class Person
{
    string nama;
    int usia, tinggi;
    double berat;

public:
    Person() {nama=""; usia=tinggi=0; berat=0.0;};
    void setPerson(string n, int u, int t, double b)
    { nama=n; usia=u; tinggi=t; berat=b; }
    string getNama() { return nama; }
    int getUsia() { return usia; }
    int getTinggi() { return tinggi; }
    void setNama(string n) { nama=n; }
};
```

Desain Class nPerson → People

Materi Praktikum 02 : Class Person

Sebagai implementasi software dari objek "beberapa person"

Memiliki atribut : **nama, usia, tinggi, berat.**

Memiliki behaviour (perilaku, methods, etc) :

- Inisialisasi (constructor)
- Dapat diberi nilai atribut (mutator)
- Dapat diakses nilai atribut (accessor)

People

-orang : Person[SIZE]
-populasi : int

+<<constructor>>People()
+getRataanTinggi()
+getLebihTinggi()
+print()

Class People

Materi Praktikum 02 : Class Person

```
class People
{
    Person orang[SIZE];
    int populasi;
public:
    People() { populasi=0; }
    void add(Person p)
    { orang[populasi++]=p; }
    void print()
    {
        for(int i=0; i<populasi; i++)
            cout << orang[i].getNama()
                << " " << orang[i].getUsia()
                << endl;
    }
    double rataanTinggi();
    int lebihTinggi();
};
```

```
double People::rataanTinggi()
{
    int sum=0;
    for(int i=0; i<populasi; i++)
        sum+=orang[i].getTinggi();
    return (double)sum/populasi;
}
```

```
int People::lebihTinggi()
{
    double r = rataanTinggi();
    int res=0;
    for(int i=0; i<populasi; i++)
        if (orang[i].getTinggi()>r)
            res++;
    return res;
}
```

Driver

Materi Praktikum 02 : Class Person

```
int main()
{
    int n;
    cin >> n;
    People mhs;
    for(int i=0; i<n; i++)
    {
        string nama;
        int usia, tinggi;
        double berat;
        Person p;
        cin >> nama >> usia >> tinggi >> berat;
        p.setPerson(nama,usia,tinggi,berat);
        mhs.add(p);
    }
    mhs.print();
    printf("%.2lf\n",mhs.rataanTinggi());
    printf("%d\n", mhs.lebihTinggi());
    return 0;
}
```

MUTATOR and ACCESSOR

- **Mutator** merupakan **fungsi** yang dapat **memberi atau mengubah nilai** atribut/data dalam class. Fungsi ini diperlukan terutama bagi data yang memiliki akses private.
- **Accessor** merupakan **fungsi** yang digunakan untuk **membaca** nilai atribut/data pada class. Fungsi ini diperlukan karena data yang akan diakses merupakan private atau protected.

CONSTRUCTOR

Constructor

adalah fungsi khusus dalam class yang akan dieksekusi terlebih dahulu saat pembuatan *instance (object)*.

Constructor biasanya digunakan untuk **inisialisasi atribut**. Misalnya memberi nilai 0 untuk usia, tinggi, dan berat.

Karakteristik Constructor

- Memiliki identitas (nama fungsi) yang sama dengan nama class.
- Tidak memiliki return type.
- Tidak dapat dipanggil dari luar class.

Problem

Bagaimana desain objek untuk contoh kasus data berikut?

Contoh kasus:

Membaca N data orang sesuai atribut yang ada dengan format CSV, dan menyimpan ke dalam array.

Contoh masukan:

```
5
Noh Jong Hyun,17,165,60.5
Yoon Ji Ho,25,170,48.7
Nam Se Hee,30,180,60.2
Ri Jung Hyuk,35,182,62.8
Yoon Se Ri,30,170,58.9
```

Pada class Person, tambahkan prosedur untuk menerima data string (satu baris data) sesuai format CSV. Di dalamnya ada parsing untuk menyimpan ke dalam setiap atribut. Misal:

```
void addPerson(string csv);
```

Class Person

Membaca data CSV

```
void Person::addPerson(string str)
{
    istringstream ss(str);
    string token;

    getline(ss, token, ','); nama=token;

    getline(ss, token, ',');
    stringstream d1(token); d1>>usia;

    getline(ss, token, ',');
    stringstream d2(token); d2>>tinggi;

    getline(ss, token, ',');
    stringstream d3(token); d3>>berat;
}
```

```
#include <iostream>
#include <cstdio>
#include <sstream>
#include <string>
#define SIZE 100
using namespace std;

class Person
{
    string nama;
    int usia, tinggi;
    double berat;

public:
    Person() {nama=""; usia=tinggi=0; berat=0.0;};
    void setPerson(string n, int u, int t, double b)
    { nama=n; usia=u; tinggi=t; berat=b; }
    string getNama() { return nama; }
    int getUsia() { return usia; }
    int getTinggi() { return tinggi; }
    void setNama(string n) { nama=n; }
    void addPerson(string csv);
};
```

Driver

Membaca data CSV

```
int main()
{
    int n;
    scanf("%d ", &n);
    People mhs;
    for(int i=0; i<n; i++)
    {
        Person p;
        string line;
        getline(cin, line);
        p.addPerson(line);
        mhs.add(p);
    }
    mhs.print();
    printf("%.2lf\n", mhs.rataanTinggi());
    printf("%d\n", mhs.lebihTinggi());
    return 0;
}
```

LATIHAN

Salah Satu Materi Praktikum 03

Sebagai latihan, lengkapi member functions dari class Person tersebut untuk hal-hal berikut:

- mengakses data berat dengan fungsi asesor `getBerat()`.
- menghitung indeks massa tubuh (IMT) menggunakan fungsi `getIMT()`, dimana nilai IMT adalah berat (kg) dibagi dengan kuadrat dari tinggi badan (meter).
- menentukan status gizi orang tersebut menggunakan fungsi `getStatusGizi()`, dimana seseorang berstatus "sangat kurus" jika $IMT < 17.0$, "kurus" jika $17.0 \leq IMT < 18.5$, "normal" jika $18.5 \leq IMT < 25.0$, "gemuk" jika $25.0 \leq IMT < 28.0$, dan "sangat gemuk" jika $IMT \geq 28.0$



IPB University
— Bogor Indonesia —

KOM120C -- BAHASA PEMROGRAMAN

Template Programming

- Function Template
- Standard Template Library

Template

Dasar untuk pemrograman generik (general programming)

Ide sederhana dalam template adalah mengirimkan tipe data sebagai parameter sedemikian rupa sehingga kita tidak perlu menuliskan kode program untuk berbagai tipe data yang berbeda.

Sebagai contoh, kita ingin dapat menentukan bilangan terbesar (maksimum) dari 2 bilangan int, juga 2 bilangan float, juga 2 bilangan double, dsb. Maka kita tidak perlu menuliskan kode untuk setiap tipe data, seperti yang kita lakukan pada saat membahas topik *function overloading*.

Template akan di-ekspansi pada saat kompilasi. Mirip *macro*. Bedanya untuk template adalah kompilator tidak memeriksa tipe data sebelum di-ekspansi.

Function Overloading

Pembeda fungsi adalah elemen prototipe fungsinya (return type, function-name, argument)

```
int maks(int x, int y)
{
    return (x > y) ? x: y;
}
double maks(double x, double y)
{
    return (x > y) ? x: y;
}
char maks(char x, char y)
{
    return (x > y) ? x: y;
}
```

```
int main()
{
    cout << maks(3,7) << endl;
    cout << maks(3.0,7.0) << endl;
    cout << maks('g','e') << endl;

    return 0;
}
```

Function Template

Fungsi generik yang dapat digunakan untuk berbagai tipe data

```
template <typename T>
T maks (T x, T y)
{
    return (x > y) ? x: y;
}

int main()
{
    cout << maks<int>(3,7) << endl;
    cout << maks<double>(3.0,7.0) << endl;
    cout << maks<char>('g','e') << endl;

    return 0;
}
```

Function Templates. Kita membuat fungsi generik yang dapat digunakan untuk berbagai tipe data.

```
template <typename T>
T maks (T x, T y)
{
    return (x > y) ? x: y;
}
```

Class Template

Mendefinisikan class yang bebas dari tipe data. Umumnya digunakan dalam class struktur data. Contoh : membuat class Array, LinkedList, Stack, Queue, etc.

```
template <typename T>
class myArray {
private:
    T *ptr;
    int size;
public:
    myArray(T arr[], int s) {
        ptr = new T[s];
        size = s;
        for(int i=0; i<size; i++)
            ptr[i] = arr[i];
    }
    void print();
};
```

```
template <typename T>
void myArray<T>::print() {
    for (int i=0; i<size; i++)
        cout<<* (ptr + i);
    cout<<endl;
}

int main() {
    int arr[5] = {1,2,3,4,5};
    myArray<int> a(arr, 5);
    a.print();
    return 0;
}
```

Template Specialization

Sifat generik dari suatu *function and class template* dapat dibuat perkecualian

```
template <typename T>
T maks(T x, T y) {
    return (x > y) ? x: y;
}

template <>
char maks(char x, char y) {
    return (x < y) ? x: y;
}
```

Template Specialization.

Fungsi `maks` didefinisikan secara generik, kecuali untuk tipe data `char`.

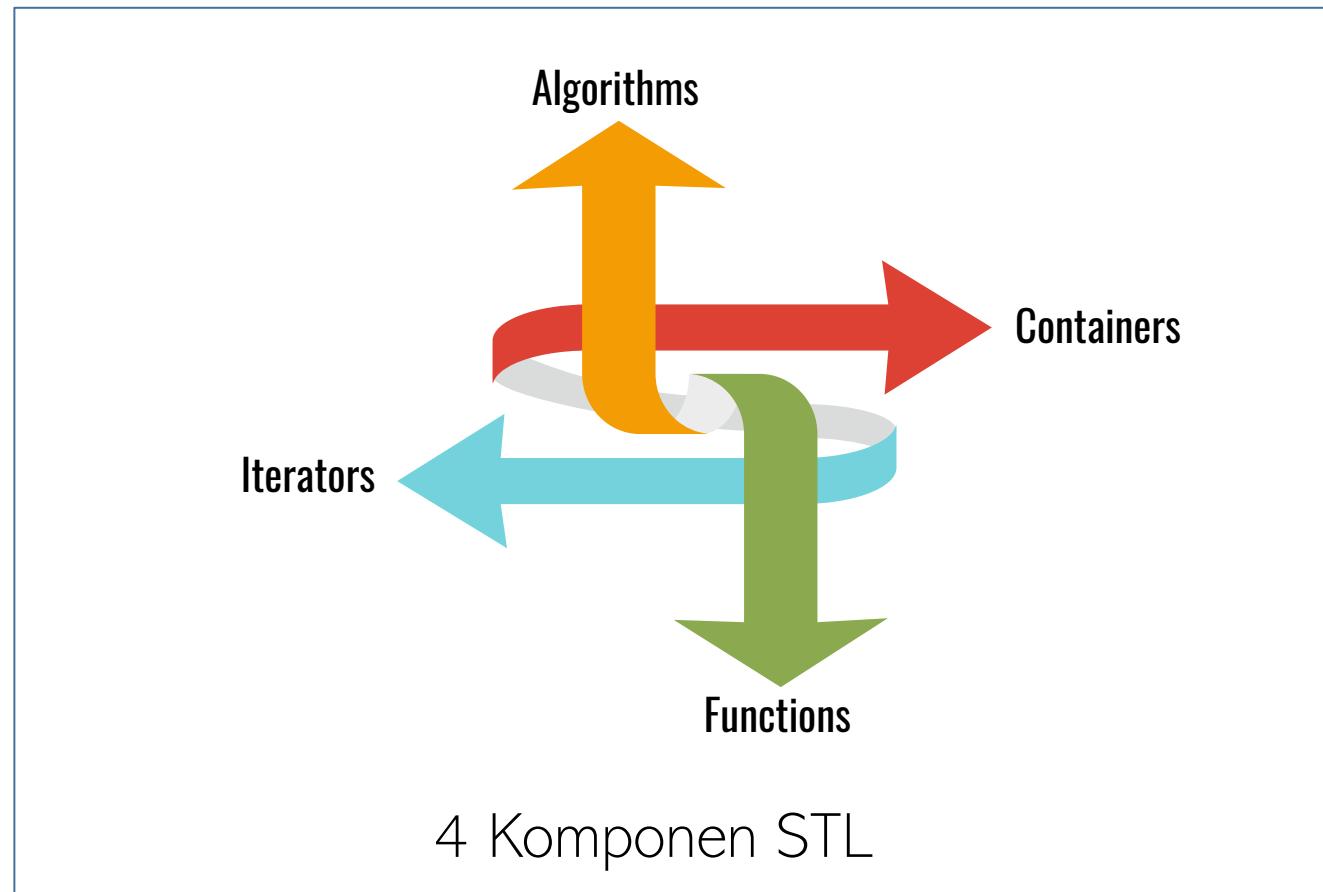
`maks(5, 10)` → 10

`maks('A', 'B')` → 'A'

Standard Template Library (STL)

Salah satu referensi → <https://www.cplusplus.com/reference/stl/>

STL adalah sekumpulan *template class* yang berisi fungsi dan struktur data untuk pemrograman generik seperti list, stack, queue, etc.



STL :: Algorithm

Komponen ini berupa **header** yang mendefinisikan sekumpulan fungsi yang dirancang khusus untuk melakukan algoritme tertentu, misalnya: sort, search, manipulasi array, dsb. Berikut salah satu contoh:

```
#include <algorithm>
#include <iostream>
int main()
{
    int a[] = { 1, 5, 8, 9, 6, 7, 3, 4, 2, 0 };
    int size = sizeof(a) / sizeof(a[0]);
    sort(a, a + size);
    ...
    return 0;
}
```

header file

panggil fungsi sort()

STL :: Container

Container (juga container class) menyimpan objek dan data. Berisi implementasi dari struktur data, seperti set, list, vector, map, stack, queue, etc. Kita akan lebih banyak membahas komponen ini.

STL :: Iterator

Iterator digunakan untuk menunjuk (sebagai pointer) ke alamat memori dari STL Container.

Operasi iterator: `begin()`, `end()`, `advance()`, `next()`, `prev()`

STL :: Function

STL juga menyediakan class yang meng-overload fungsi. Instance dari class ini disebut sebagai objek fungsi atau functors (bukan fungsi). Contoh kegunaan: program berikut menambah 1 ke setiap elemen array.

```
int increment(int x) {    return (x+1); }

int main()
{
    int arr[] = {1, 2, 3, 4, 5};
    int n = sizeof(arr)/sizeof(arr[0]);

    transform(arr, arr+n, arr, increment);
    // ...

    return 0;
}
```

The code snippet shows a C++ program. It defines a function `increment` that takes an integer `x` and returns `(x+1)`. The `main` function initializes an array `arr` with values 1, 2, 3, 4, 5, calculates its size `n`, and then uses the `transform` function to iterate over the array, applying the `increment` function to each element. The result is stored back in the array. The `transform` call is highlighted with a red box, and a red arrow points from it to a red box containing the signature of the `transform` function: `transform(inputBegin, inputEnd, OutputBegin, unary_operation)`.

STL :: set

Himpunan nilai atau objek unik dengan tipe data tertentu. Nilai disimpan dalam urutan tertentu.

STL juga menyediakan class yang meng-overload fungsi. Instance dari class ini disebut sebagai objek fungsi atau functors (bukan fungsi). Contoh kegunaan: program berikut menambah 1 ke setiap elemen array.

Sintaks:

```
#include <set>
set<datatype> setname;
```

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

int main()
{
    set<int> s1;
    s1.insert(40); s1.insert(30);
    s1.insert(60); s1.insert(20); s1.insert(40);

    set<int>::iterator p;
    for(p=s1.begin();p!=s1.end();++p)
        cout << *p << " ";
    cout << endl;
    return 0;
}
```

Contoh:

```
set<int> s1; // empty set
set<int> s2 = { 6, 10, 5, 1 }; // disimpan {1,5,6,10}
```

Output:

20 30 40 60

Bagaimana jika ingin urutan **descending**?

Gunakan unary operation function:

```
set<int, greater<int> > s1;
```

STL :: vector

Himpunan nilai atau objek dengan tipe data tertentu. Sama dengan array dinamis.

STL juga menyediakan class yang meng-overload fungsi. Instance dari class ini disebut sebagai objek fungsi atau functors (bukan fungsi). Contoh kegunaan: program berikut menambah 1 ke setiap elemen array.

Sintaks:

```
#include <vector>
vector<datatype> vectornname;
```

Contoh:

```
vector<int> v;                      // empty vector
v.assign(7, 100);                    // 7 elemen bernilai 100
v.push_back(15);                   // menambah elemen bernilai 15
cout << v.size();                  // 8

// print semua elemen
vector<int>::iterator it; // create iterator
for(it=v.begin(); it!=v.end(); ++it)
    cout << *it;
```

Vector of Class

Bisa dilakukan karena class merupakan ADT. Hal yang sama berlaku pada STL lainnya.

Contoh kasus: class People yang merupakan objek dari n Person.

```
class People
{
    Person orang[SIZE];
    int populasi;
public:
    People() { populasi=0; }
    void add(Person p)
    { orang[populasi++]=p; }
    void print();
    double rataanTinggi();
    int lebihTinggi();
};
```

```
class People
{
    vector<Person> orang;
    int populasi;
public:
    People() { populasi=0; }
    void add(Person p)
    { orang.push_back(p); populasi=orang.size(); }
    void print()
    {
        vector<Person>::iterator p;
        for(p=orang.begin(); p!=orang.end(); ++p)
        {
            cout << p->getNama()
                << " " << p->getUsia() << endl;
        }
    }
    double rataanTinggi(); // bagaimana ?
    int lebihTinggi(); // bagaimana ?
};
```

STL :: pair

Merupakan salah satu utility dalam STL C++.

Sintaks:

```
#include <utility>
pair<datatype, datatype> pairname;
```

Contoh:

```
pair<int, int> p1;
p1.first = 100;  p1.second = 76;
pair<string, Person> p2;          // Person adalah class
                                // Misalnya string untuk menyimpan NIK
                                // sebagai primary key
```

Latihan

Gunakan konsep OOP

Deskripsi

Buat program membaca beberapa bilangan bulat, dan menghapus beberapa bilangan pada posisi tertentu.

Format Masukan

Bagian pertama berisi beberapa bilangan bulat yang diakhiri dengan nilai -9 (sentinel). Bagian kedua adalah beberapa bilangan terurut dari kecil ke besar yang menunjukkan elemen keberapa dari bilangan masukan yang dihapus. Diakhiri dengan -9 (sentinel).

Format Keluaran

Baris pertama adalah dua bilangan yang menunjukkan banyaknya bilangan awal dan banyaknya bilangan setelah dihapus. Baris kedua adalah dua nilai rata-rata yang dituliskan dalam dua digit di belakang tanda desimal dari bilangan-bilangan awal dan bilangan-bilangan setelah dihapus. Jika data dalam array kosong, maka nilai rata-rata dituliskan -9.99.

Contoh Input

10 20 30 40 50 -9 1 3 4 -9

Contoh Output

5 2
30.00 35.00



IPB University
— Bogor Indonesia —

KOM120C -- BAHASA PEMROGRAMAN

Template Programming #2

- STL (Lanjutan)
- Operator Overloading

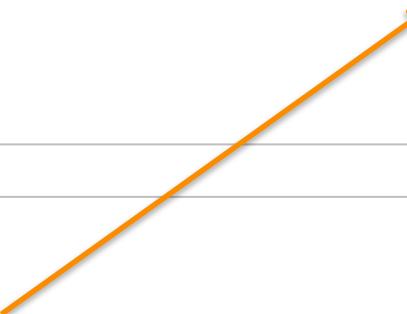
STL :: map

Himpunan elemen yang terdiri atas **key** dan **value**. Nilai disimpan dalam urutan tertentu dari key. Merupakan **associative array**.

Sintaks:

```
#include <map>
map<keytype, valuetype> mapname;
```

```
st["G6421002"] = "Ji Pyeong";
st["G6421005"] = "Siapa Saja";
```



Contoh:

```
map<string, string> st;
st.insert(pair<string, string>("G6421005", "Siapa Saja"));
st.insert(pair<string, string>("G6421002", "Ji Pyeong"));
st.insert(pair<string, string>("G6421002", "Gundala Putra Petir"));

map<string, string>::iterator p;
for(p=st.begin(); p!=st.end(); ++p)
    cout << p->first << " " << p->second << endl;

for(const auto& ptr : st)
    cout << ptr.first << " --> " << ptr.second << endl;
```

STL :: map

Bagaimana jika value berupa class.

Contoh Problem:

Diinginkan untuk menyimpan data pegawai yang menggunakan class Person.

Setiap pegawai memiliki ID yang unik. Contoh data:

```
"3204", "Siapa Saja", 17, 170, 72.5  
"3201", "Ji Pyeong", 19, 180, 70.2
```

```
map<string, Person> st;  
Person p;  
p.setPerson("Siapa Saja", 17, 170, 72.5);  
st.insert(pair<string, Person>("3204", p));  
  
p.setPerson("Ji Pyeong", 19, 180, 70.2);  
st.insert(pair<string, Person>("3201", p));  
  
map<string, Person>::iterator ptr;  
for(ptr=st.begin(); ptr!=st.end(); ++ptr)  
    cout << ptr->first << " --> " << (ptr->second).getNama() << endl;
```

3201 --> Ji Pyeong

3204 --> Siapa Saja



STL :: map

Bagaimana menyusun map secara terbalik atau mengikuti logika tertentu?

STL Map tidak memiliki fitur descending order.

Prosedur:

- Pindahkan struktur MAP ke struktur VECTOR yang memiliki algoritme SORT.
- Gunakan algoritme SORT dengan menggunakan logika pembandingan tertentu (compare).

```
// descending logics
bool cmp(pair<string, Person>& a, pair<string, Person>& b)
{
    return a.first > b.first;
}

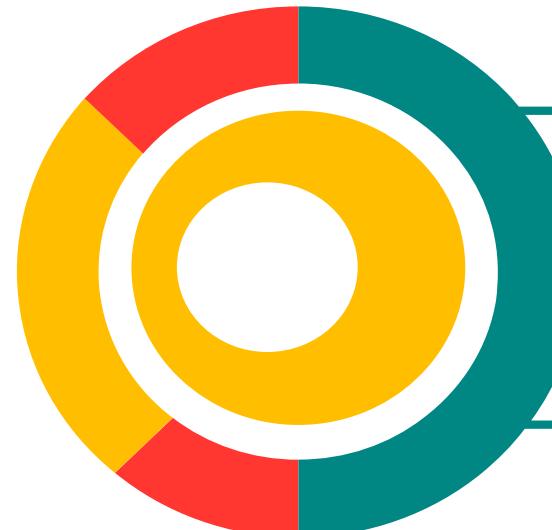
void sort(map<string, Person>& M)
{
    vector<pair<string, Person> > A;

    for (auto& it : M) {                                // copy map to vector
        A.push_back(it);
    }
    sort(A.begin(), A.end(), cmp);

    for (auto& it : A) {                                // print all elements
        cout << it.first << " --> "
            << (it.second).getNama() << endl;
    }
}
```

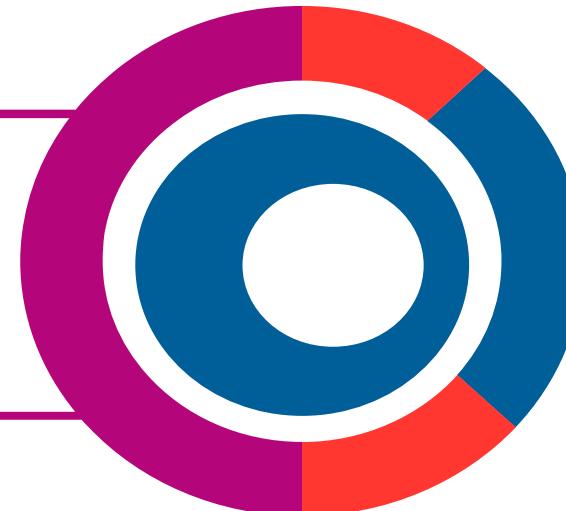
Overloading

C++ memungkinkan untuk membuat satu atau lebih definisi untuk nama fungsi atau nama operator dalam lingkup yang sama.



Function Overloading

Fitur dalam OOP yang mengizinkan 1 atau lebih definisi berbeda dari suatu fungsi dengan nama yang sama.



Operator Overloading

Fitur dalam OOP yang mengizinkan 1 atau lebih definisi berbeda dari suatu operator dengan nama yang sama.

Bagaimana function overloading bekerja?



Exact match

Nama fungsi dan argumen sesuai → EXECUTE !



Promoted to Appropriate Type

char, unsigned char, short → int
float → double



Standard Conversion

Konversi baku C++ dari tipe tertentu ke tipe lain yang bersesuaian.

Contoh: Integer conversion: unsigned char,unsigned short int, unsigned int, unsigned long int, unsigned long long int.

else → ERROR!

Operator Overloading

C++ memungkinkan kita membuat fungsi operator sesuai dengan definisi yang dikehendaki.

Contoh: operator penjumlahan (+) dapat digunakan untuk menjumlahkan berbagai data (int, float, double, char), bahkan untuk menjumlahkan objek dari class (String), atau objek dari class yang dibuat oleh user.

Caranya?

Suatu operator X dapat dituliskan sebagai fungsi dengan prototipe sebagai berikut:

```
<return type> operatorX (<argument list>)
```

Contoh Kasus

Terdapat class Complex untuk mengimplementasikan objek bilangan complex yang terdiri atas 2 bagian data, yaitu bilangan real dan imaginer. Objek dapat diolah menggunakan operator aritmatika +.

```
class Complex
{
private:
    int real, imaginer;
public:
    Complex(int r=0, int i=0) { real=r; imaginer=i; }
    Complex operator+ (Complex const &c) {
        Complex res;
        res.real = real + c.real;
        res.imaginer = imaginer + c.imaginer;
        return res;
    }
    void print() {
        cout << real << " + " << imaginer << endl;
    }
};
```

```
int main()
{
    Complex c1(10,5), c2(2,4);
    Complex c3, c4;
    c3=c1+c2;
    c4=c1.operator+(c2);
    c1.print();
    c2.print();
    c3.print();
    c4.print();

    return 0;
}
```

Prefix and Postfix Operator Overloading

C++ memiliki 2 jenis increment/decrement operator:

- Prefix increment and postfix increment
- Prefix decrement and postfix decrement

Implementasi untuk Class Complex (contoh):

```
Complex& operator++();           // Prefix increment  
Complex operator++(int);         // Postfix increment  
  
Complex& operator--();           // Prefix decrement  
Complex operator--(int);         // Postfix decrement
```

Latihan

Gunakan konsep OOP

Deskripsi

Buat program mengelola bilangan pecahan a/b/c.

Format Masukan

Beberapa baris operasi bilangan pecahan:

set a b c	inisialisasi bilangan pecahan a/b/c
p	menampilkan bilangan pecahan sesederhana mungkin
add a b c	menambah bilangan pecahan yang ada dengan a/b/c
mul a b c	mengalikan bilangan pecahan yang ada dengan a/b/c
inc	postfix increment
dec	postfix decrement
end	akhir dari pengolahan

Format Keluaran

Beberapa baris bilangan pecahan sesederhana mungkin.

Contoh Input

```
set 4 2 8  
p  
add 0 6 8  
inc  
p
```

Contoh Output

```
4 1/4  
5
```



IPB University
— Bogor Indonesia —

KOM120C -- BAHASA PEMROGRAMAN

Inheritance

- Inheritanca (Pewarisan)
 - Overidding
-

Tim Pengajar Bahasa Pemrograman IPB University

Class Person

Implementasi dari objek orang.

Class Person ingin digunakan untuk implementasi objek Mahasiswa

```
class Person {  
    private:  
        string name;           // nama  
        string address;        // alamat  
        int age;               // usia  
    public:  
        Person() { name=address=""; age=0; }  
        Person(string nm, String ad, int th) { name=nm; address=ad; age=th; }  
        void setName(string nm) { name=nm; }  
        void setAddress(string ad) { address=ad; }  
        void setAge(int th) { age=th; }  
        string getName() { return name; }  
        string getAddress() { return address; }  
        int getAge() { return age; }  
        void print() { cout << name << ", " << address << age << endl; }  
};
```

Kebutuhan Baru

Class Person ingin digunakan untuk implementasi objek Mahasiswa

Beberapa problem:

- Mahasiswa merupakan “person” yang memiliki atribut tambahan berupa (contoh) nim dan daftar MK yang telah diambil
- Mahasiswa dapat meng-update MK yang diambil.

Class Person harus di-EXTEND → buat class Student yang **mewarisi** sifat-sifat class Person → **pewarisan/inheritance**.

Class Student sebagai TURUNAN dari class Person → **DERIVED CLASS**.

Class Person sebagai INDUK dari class Student → **BASE CLASS**.

Class Person

Beberapa sifat harus dapat DIWARISKAN → protected and public member class

```
class Person {  
    protected:          // dapat diakses oleh class ini beserta turunannya  
    string name;  
    string address;  
    int age;  
public:  
    Person() { name=address=""; age=0; }  
    Person(string nm, String ad, int th) { name=nm; address=ad; age=th; }  
    void setName(string nm) { name=nm; }  
    void setAddress(string ad) { address=ad; }  
    void setAge(int th) { age=th; }  
    string getName() { return name; }  
    string getAddress() { return address; }  
    int getAge() { return age; }  
    void print() { cout << name << ", " << address << age << endl; }  
};
```

Class Student

Beberapa atribut dan prosedur mengambil dari sifat induknya (Person)

```
class MK {                                // implementasi dari struct MK
public:
    string kodemk;
    int sks;
};
```

```
class Student : public Person {
private:
    string nim;
    vector<MK> krs;
public:
    Mahasiswa() { nim=""; }
    setMahasiswa(string nm, string ad, int th, string n)
        { name=nm; address=ad; age=th; nim=n; }
    void setKRS() { .... }
    .... dst.
};
```

Kaidah Pewarisan

Pada prinsipnya, setiap anggota base class diturunkan ke derived class. Hanya tingkatan aksesnya yang berbeda, tergantung jenis penurunannya.

Perkecualian: anggota private (tidak diturunkan), default constructor (otomatis).

		Jenis Penurunan			
Untuk anggota		private	protected	public	
Kontrol akses	private	-	-	-	
	protected	private	protected	protected	
	public	private	protected	public	

Mengapa Pewarisan ?

Pewarisan adalah mekanisme untuk:

- Mengembangkan class baru dari class yang sudah ada
- Mendefinisikan class baru sebagai spesialisasi dari class yang sudah ada.

Harus memahami istilah base class dan derived class !

Berikan contoh ...

Mendefinisikan Pewarisan

Sintaks:

```
class <DerivedClass> : <access-level> <BaseClass>
```

<access-level> menunjukkan tipe pewarisan

- private (by default)
- public

Setiap class dapat berfungsi sebagai Base Class, sehingga Derived Class dapat dibuat sebagai Base Class. Memungkinkan turunan dari class turunan.

Constructor untuk Turunan

Default constructor dan destructor dari base class selalu dipanggil ketika suatu objek baru dari class turunannya dibuat atau di-destroy.

```
class A {  
public:  
A() {  
cout<<"A:default\n"; }  
A(int a) {  
cout<<"A:parameter\n"; }  
};
```

```
class B : public A {  
public:  
B(int a) {  
cout<<"B\n"; }  
};
```

B x(1);



output:
A:default
B

Constructor untuk Turunan

Kita dapat menentukan constructor dari base class yang digunakan untuk class turunannya.

```
class A {  
public:  
A() {  
cout<<"A:default\n"; }  
A(int a) {  
cout<<"A:parameter\n"; }  
};
```

```
class C : public A {  
public:  
C (int a) : A(a) {  
cout<<"C"<<endl; }  
};
```

C x(1);



output:

A:parameter
C

Overriding

Class turunan dapat meng-override fungsi yang telah didefinisikan oleh base class.

Dengan overriding,

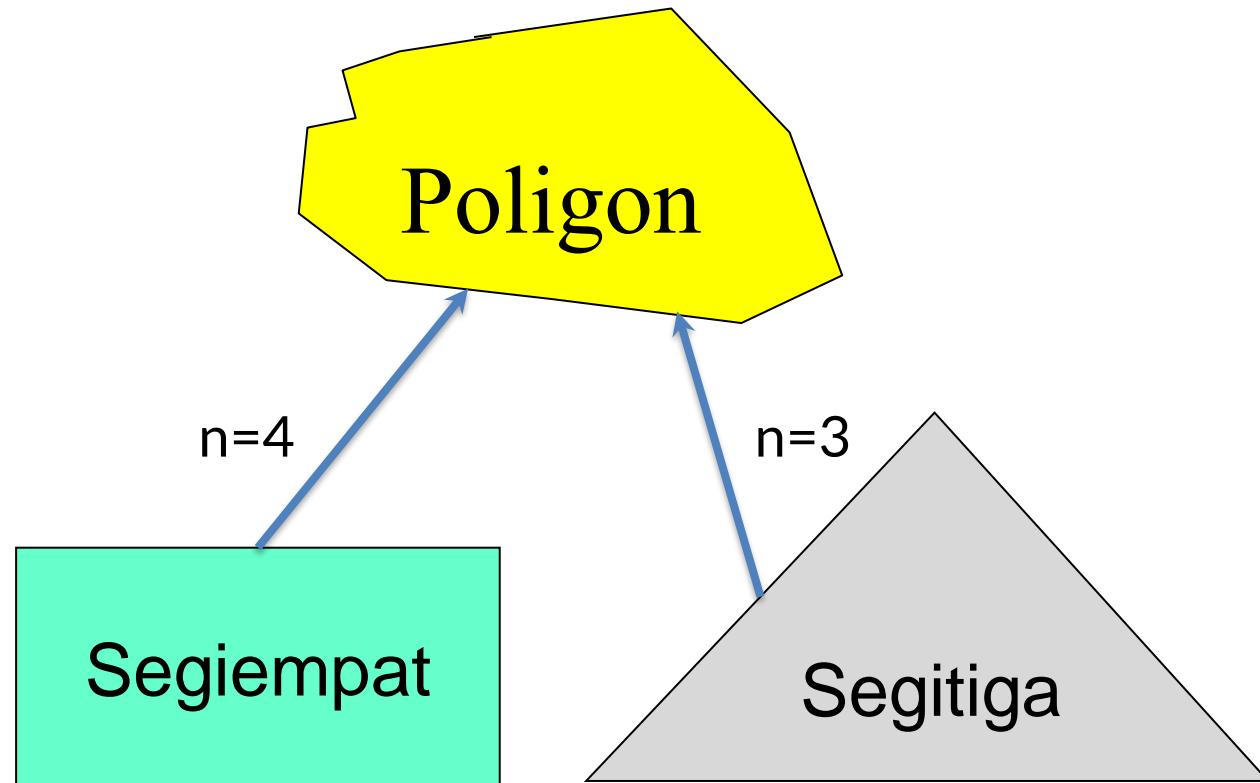
- Fungsi atau method dalam class turunan memiliki signature yang identik dengan method pada base class.
- Derive class mengimplementasikan method versinya sendiri.

```
class A {  
protected:  
    int x, y;  
public:  
    void print(){  
        cout<<"A"<<endl; }  
};
```

```
class B : public A {  
public:  
    void print() {  
        cout<<"B"<<endl; }  
};
```

Contoh Pewarisan #1 :: Spesialisasi

Implementasi dari objek Poligon, Segiempat, dan Segitiga



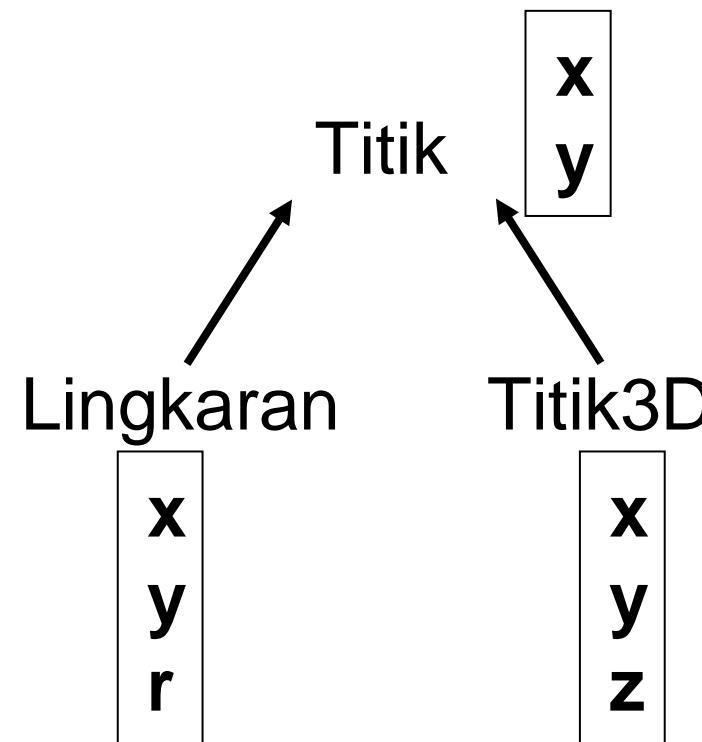
```
class Poligon {  
protected:  
    int n;  
    vector <double> x, y;  
public:  
    void set(...);  
};
```

```
class Segiempat:public Poligon {  
public:  
    double luas();  
};
```

```
class Segitiga:public Poligon {  
public:  
    double luas();  
};
```

Contoh Pewarisan #2 :: Extensification

Implementasi dari objek Titik 2D, Titik 3D, dan Lingkaran



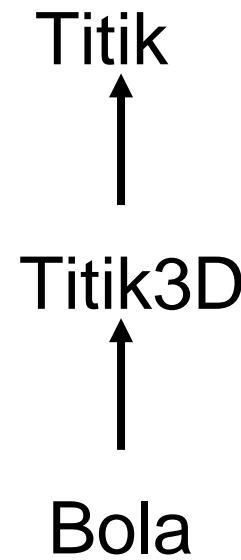
```
class Titik {
protected:
    int x, y;
public:
    void set (int a, int b);
};
```

```
class Lingkaran :
public Titik {
    double r;
};
```

```
class Titik3D :
public Titik {
    int z;
};
```

Contoh Pewarisan #3 :: Turunan dari Turunan

Titik adalah baseclass dari Titik3D, sedangkan Titik3D adalah baseclass dari Bola



```
class Titik {  
protected:  
    int x, y;  
public:  
    void set (int a, int b);  
};
```

```
class Titik3D : public Titik {  
private:  
    double z;  
    ...  
};
```

```
class Bola : public Titik3D {  
private:  
    double r;  
    ...  
};
```

Latihan

Gunakan konsep OOP

Deskripsi

[Praktikum-06-A-PewarisanPegawai.pdf](#)

Format Masukan

[N : banyaknya pegawai, $1 < N < 100$]

[N baris data pegawai : id, usia, tipe. Jika pegawai tetap, ada nilai gaji pokok.]

[T baris data gaji : id dan upah (pegawai harian) | uang lembur (pegawai tetap).]

END

Format Keluaran

Beberapa baris data id, tipe, dan penghasilan sebulan. Kelompok pegawai tetap di bagian atas, disambung dengan kelompok pegawai harian. Urutan data sesuai dengan urutan data masuk.

Contoh Input

```
5
123456 19 1 5000
989212 21 1 6000
876523 20 2
092831 20 2
187632 19 1 5000
123456 2000
876523 1000
092831 5000
187632 4000
END
```

Contoh Output

```
123456 1 7000
989212 1 6000
187632 1 9000
876523 2 1000
092831 2 5000
```



KOM120C -- BAHASA PEMROGRAMAN

Inheritance (continue)

- Type of Inheritance in C++
 - Static Keyword
 - Virtual Function
 - Abstract Class
-

Jenis Pewarisan dalam C++

Single Inheritance

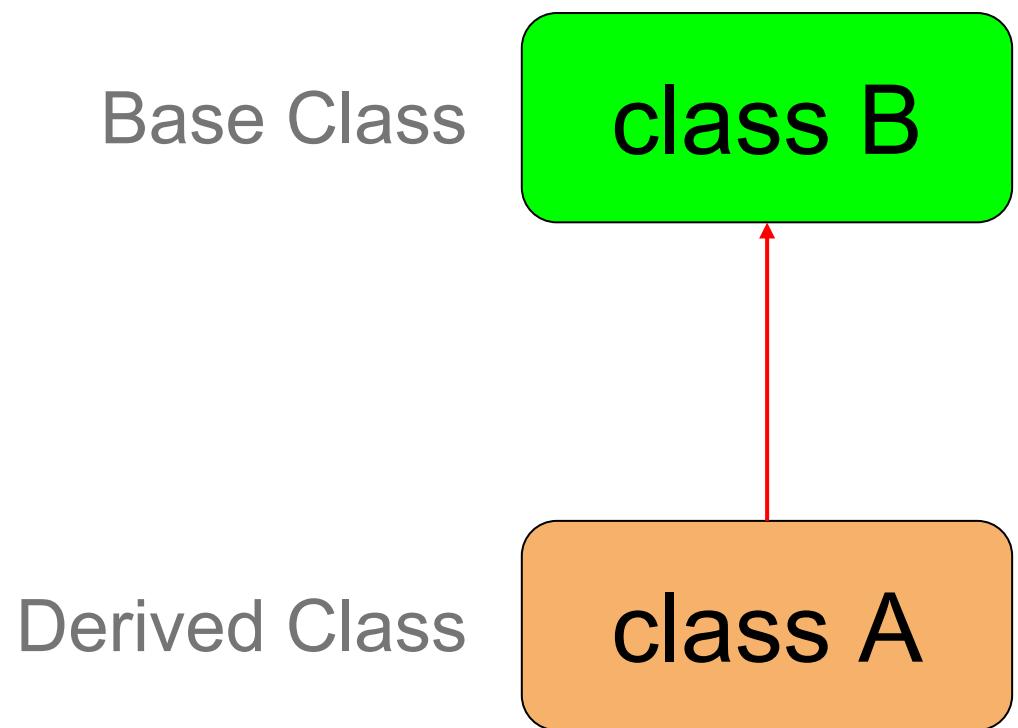
Multiple Inheritance

Multilevel Inheritance

Hierarchical Inheritance

Single Inheritance

Base class hanya memiliki sebuah derived class



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

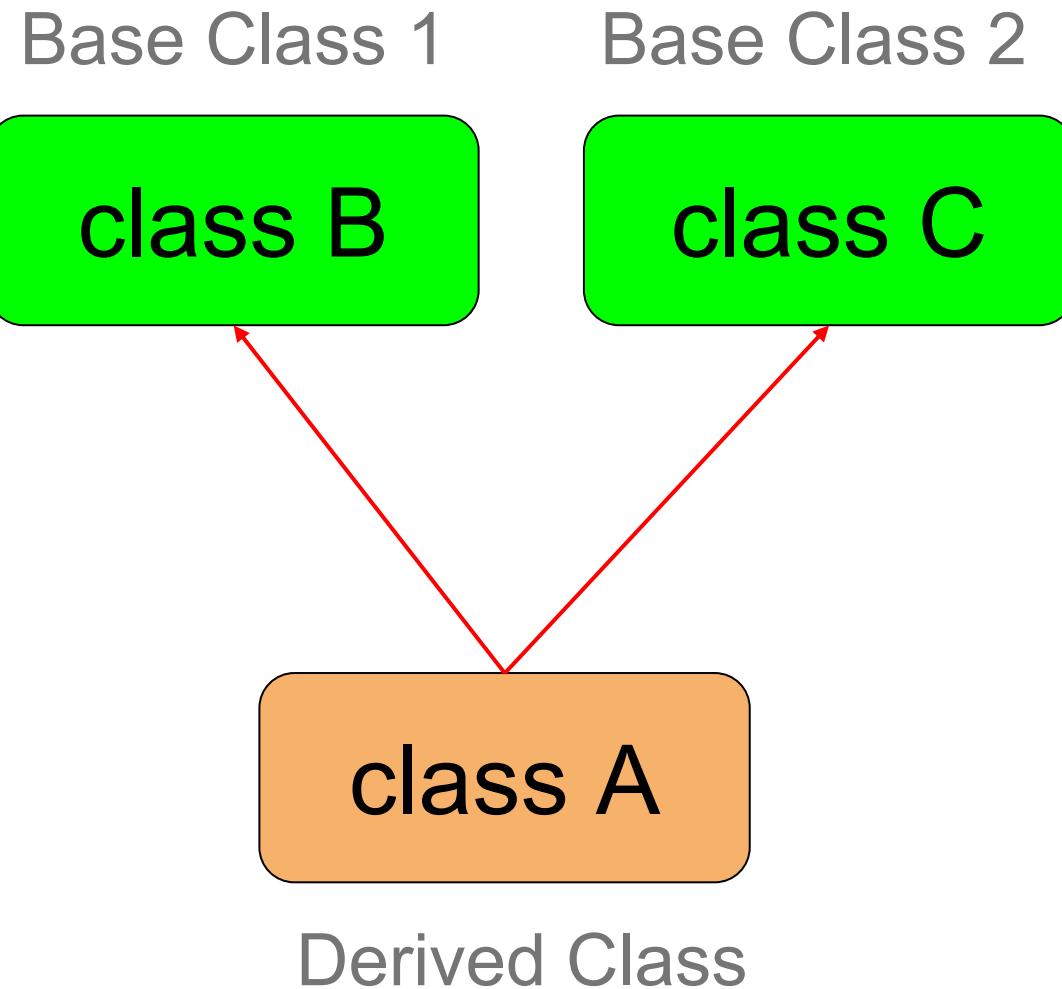
class Kendaraan {
public:
    Kendaraan() { cout << "Ini kendaraan\n"; }

class Mobil : public Kendaraan {
    Mobil() { cout << "Ini mobil\n"; }

int main()
{
    Mobil obj;
    return 0;
}
```

Multiple Inheritance

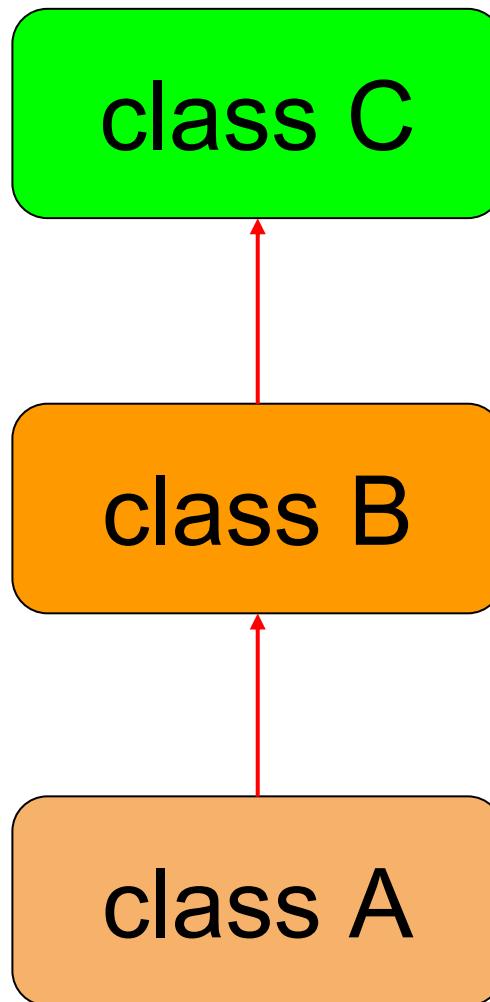
Sebuah derived class diturunkan lebih dari satu base class



```
class Kendaraan {  
public:  
    Kendaraan() { cout << "Ini kendaraan\n"; }  
};  
  
class Roda4 {  
public:  
    Roda4() { cout << "Ini kendaraan roda 4\n"; }  
};  
  
class Mobil : public Kendaraan, public Roda4 {  
    Mobil() { cout << "Ini mobil\n"; }  
};  
  
int main()  
{  
    Mobil obj;  
    return 0;  
}
```

Multilevel Inheritance

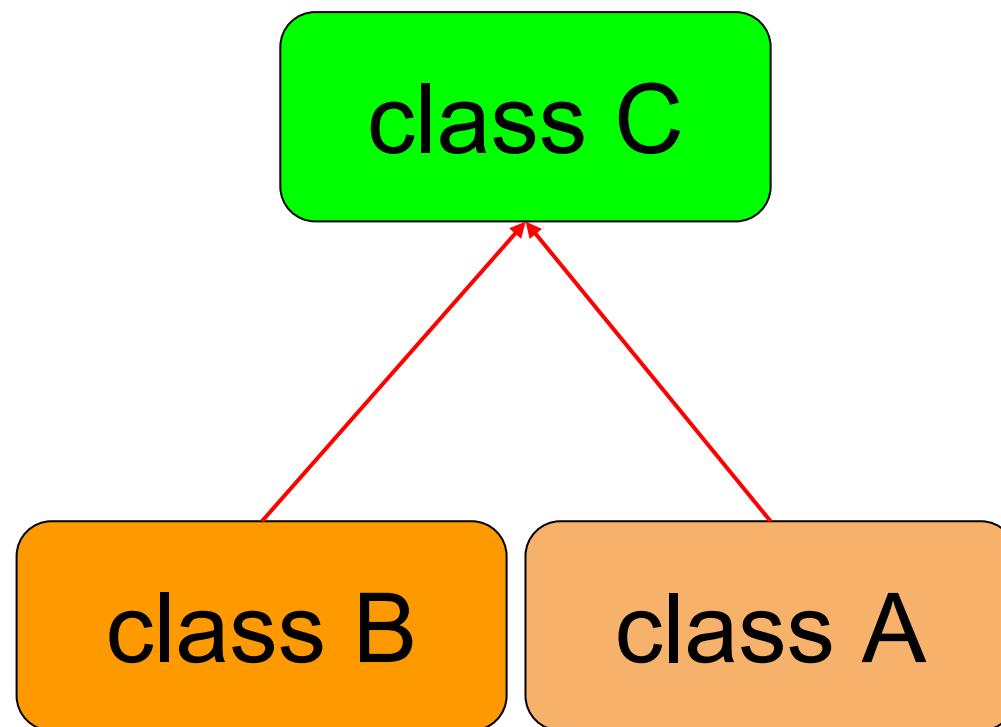
Sebuah derived class menjadi base class



```
class Kendaraan {  
public:  
    Kendaraan() { cout << "Ini kendaraan\n"; }  
};  
  
class Roda4 : public Kendaraan {  
public:  
    Roda4() { cout << "Ini kendaraan roda 4\n"; }  
};  
  
class Mobil : public Roda4 {  
Mobil() { cout << "Ini mobil\n"; }  
};  
  
int main()  
{  
    Mobil obj;  
    return 0;  
}
```

Hierarchical Inheritance

Lebih dari satu derived class diturunkan dari sebuah base class



```
class Kendaraan {  
public:  
    Kendaraan() { cout << "Ini kendaraan\n"; }  
};  
  
class Sedan : public Kendaraan {  
    Mobil() { cout << "Ini sedan\n"; }  
};  
  
class Bus : public Kendaraan {  
    Bus() { cout << "Ini Bus\n"; }  
};  
  
int main()  
{  
    Sedan obj1;  
    Bus obj2;  
    return 0;  
}
```

Static Keyword in C++

- Static variables
 - Static members of class
-

Static Variables in a Function

- Ketika variabel dideklarasikan sebagai static, maka alokasi variabel tersebut berlaku selama program berjalan.
- Artinya, alokasi untuk variabel static di dalam fungsi akan dilakukan hanya sekali walaupun fungsi tersebut dipanggil berkali-kali.

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

void demo()
{
    static int c = 4;
    cout << c << " ";
    c++;
}

int main()
{
    for (int i=0; i<5; i++)
        demo();
    return 0;
}
```

Static Variables in a Class

- Ketika variabel dideklarasikan sebagai static dalam class, maka alokasi variabel tersebut akan di-share oleh objek.
- Artinya, variabel static tidak dapat disalin untuk objek yang berbeda.
- Variabel static tidak dapat diinisialisasi menggunakan constructor.

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

class myClass
{
public:
    static int i;
    myClass() { }
};

int main()
{
    myClass obj1;
    myClass obj2;
    obj1.i = 2;
    obj2.i = 3;

    cout << obj1.i << " " << obj2.i;
    return 0;
}
```

```
class myClass
{
public:
    static int i;
    myClass() { }

};

int myClass::i = 4;

int main()
{
    myClass obj;
    cout << obj.i;
    return 0;
}
```

Class Objects as Static

Seperti variabel, objek yang dideklarasikan static berlaku selama program berjalan

```
// Tanpa static
#include<iostream>

class myClass
{
    int i;
public:
    myClass() {
        i = 0;
        std::cout << "A\n";
    }
    ~myClass() { cout << "B\n"; }
};

int main()
{
    int x = 0;
    if (x==0) myClass obj;
    std:: cout << "End of main\n";
    return 0;
}
```

```
// Dengan static
#include<iostream>

class myClass
{
    int i = 0;
public:
    myClass() {
        i = 0;
        std::cout << "A\n";
    }
    ~myClass() { cout << "B\n"; }
};

int main()
{
    int x = 0;
    if (x==0) { static myClass obj; }
    std::cout << "End of main\n";
    return 0;
}
```

Static Functions in a Class

- Sama seperti anggota data static atau variabel static di dalam kelas, fungsi anggota static juga tidak bergantung pada objek kelas.
- Diizinkan untuk memanggil fungsi anggota statis menggunakan objek dan operator '.', tetapi disarankan menggunakan nama kelas dan operator '::'.

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

class myClass
{
public:
    static void print()
    {
        cout<<"Welcome!";
    }
};

int main()
{
    myClass::print();
    return 0;
}
```

Object of Self Type

Class dapat mengandung object dengan tipe dirinya sendiri, namun harus static atau pointer.

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

class Test {
    static Test self;

};

int main()
{
    Test t;
    return 0;
}
```

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

class Test {
    Test *self;

};

int main()
{
    Test t;
    return 0;
}
```

Virtual Function

Virtual Functions

- Fungsi virtual adalah fungsi anggota yang dideklarasikan dalam base class dan didefinisikan ulang (ditimpa) oleh kelas turunannya.
- Fungsi virtual tidak dapat static.
- Fungsi virtual harus diakses menggunakan pointer.
- Prototipe fungsi virtual harus sama, di dalam base class dan derived class.

```
class A {  
public:  
    virtual void print() { cout << "Print A\n"; }  
    void show() { cout << "Show A\n"; }  
};  
  
class B : public A {  
public:  
    void print() { cout << "Print B\n"; }  
    void show() { cout << "Show B\n"; }  
};  
  
int main()  
{  
    A *ptr;  
    B d;  
    ptr = &d;  
    ptr->print();  
    ptr->show();  
    return 0;  
}
```

Abstract Function

Abstract Class

Abstract Function & Abstract Class

Dalam C++, abstract function disebut juga sebagai **pure virtual function**, yaitu virtual function yang harus di-override di dalam derived class.

Abstract function dapat dideklarasikan dengan memberikan assignment nilai 0.

```
class Ruang2D
{
public:
    virtual double luas() = 0;

};
```

Class yang sedikitnya mengandung sebuah abstract function disebut sebagai **abstract class**.

Latihan

Abstract Class

Ruang2D

Segiempat

Segitiga

Diketahui 3 class, yaitu Ruang2D (untuk mengolah data ruang 2 dimensi yang memiliki maksimum 4 sisi), Segiempat (untuk mengolah data segi empat), dan Segitiga (untuk mengolah data segitiga siku-siku). Class Ruang2D merupakan abstract class yang memiliki atribut 4 sisi bilangan bulat (integer), dan fungsi abstrak luas() dan keliling(), serta sebuah fungsi mutator set(). Lengkapi setiap class dan buat program OOP untuk membaca m sisi (panjang dan lebar) segiempat, dan n sisi (alas dan tinggi) segitiga siku-siku. Program menampilkan luas dan keliling segiempat diikuti luas dan keliling segitiga, terurut dimulai dari luas terbesar. Jika luas sama, data diurutkan berdasarkan keliling descending.

```
class Ruang2D
{
    protected:
        double sisil, sisi2, sisi3, sisi4;
public:
    virtual double luas() = 0;
    virtual int keliling() = 0;
    void set(int s1=0,int s2=0,
             int s3=0,int s4=0);
};
```

Contoh Input:

```
2 3          // 2 segiempat, 3 segitiga
2 3
4 5
1 2
3 4
2 3
```

Contoh Input:

```
SEGIEMPAT
20.0 18.0
6.0 10.0
SEGITIGA
6.0 13.0
3.0 8.6
1.0 5.2
```