



Tutorial 1

Praktikum Pemrograman Berbasis Objek

Asisten IF2210 2022/2023

kenalan yuk

kemarin banyak komentar *gak* kenal asisten.

Asisten IF2110 2022/2023



Viel
Koordinator Asisten
PBO 22/23



Tito
Asisten



Abi
Asisten



Rei
Asisten

Terculik ke Labpro

Asisten IF2110 2022/2023



Wiwid
Asisten

Ex-Koordinator Asisten WBD 22/23



Fabian
Asisten

Ex-Koordinator Asisten Dasar
Pemrograman 21/22



Adit
Asisten

Login PC

Username: labdas

Password: praktikum

Username: lecture...

Password: koica2006!

Outline

1. Review Praktikum Minggu Sebelumnya
2. Mekanisme Tutorial dan Praktikum
3. Praktikum Luring
4. Ctor, Cctor, Dtor
5. Const, Static
6. Friends
7. Operator Overloading
8. Function Overloading

Review Praktikum Minggu Sebelumnya

Ga ada, kan tutorial pertama.

Tutorial dan Praktikum

1. Tutorial dan praktikum dilakukan bergantian setiap minggu
2. Tutorial/praktikum dilakukan selama 120 menit
3. Saat tutorial, akan ada 30 menit berisi materi dan 90 menit mengerjakan soal. Kalau tidak selesai, bisa lanjut di rumah, deadline di hari yang sama 21.00 WIB. Boleh juga langsung keluar ruangan jika tidak yang ingin ditanyakan.
4. Saat tutorial, mahasiswa boleh bertanya/diskusi pada asisten dan browsing sepantasnya saat 120 menit selama praktikum.
5. Saat praktikum, akan ada 120 menit untuk mengerjakan soal.
6. Saat praktikum, mahasiswa tidak boleh bertanya (kecuali masalah urgent/teknis) maupun browsing.
- ~~7. Mulai minggu depan, pastikan tidak ada yang salah masuk lab yang telah dialokasikan~~

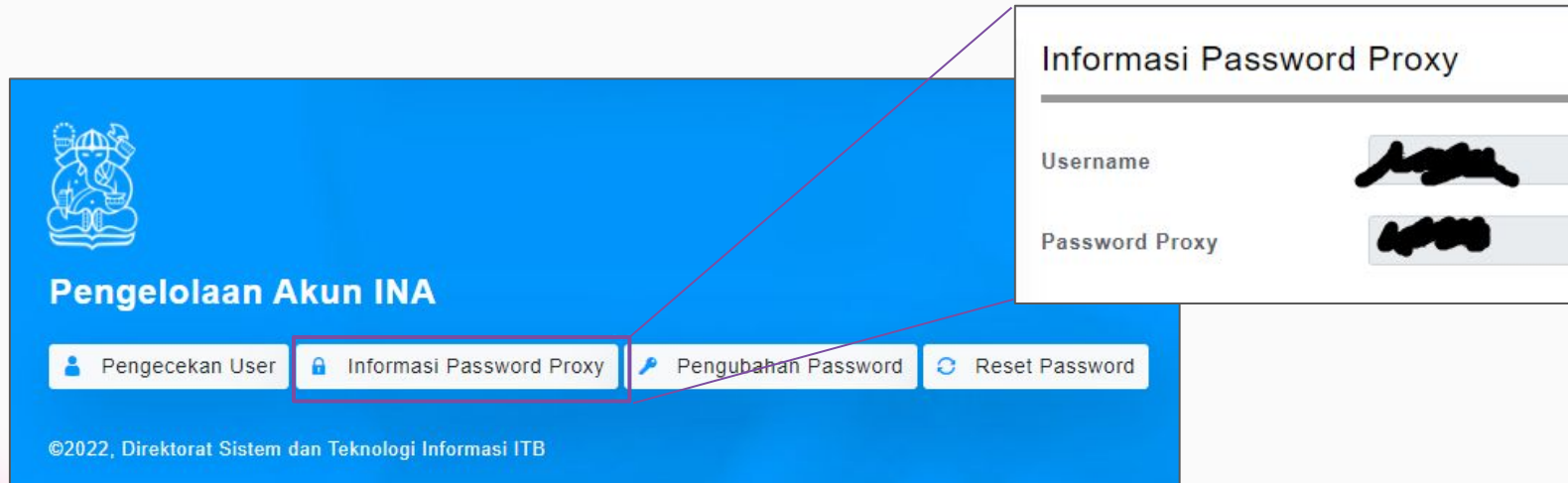
Praktikum Luring

1. Jaga kebersihan :))
2. Jaga tata tertib :)))
3. Baca dan ikuti semua peraturan yang tertera di lab
4. Kerjakan semua kegiatan dalam lab pada komputer yang tersedia
5. Beberapa lab memiliki komputer yang dual boot (tersedia Windows dan OS lain berbasis Linux), gunakan Linux.
6. Sedangkan lab lain memiliki komputer yang hanya memiliki OS berbasis linux, biasakan penggunaannya dalam tutorial ini
7. Terdapat 2 daftar hadir, daftar hadir lab dan *log book* jangan lupa untuk mengisi presensi di kedua daftar hadir tersebut

Praktikum Luring: Proxy

Jika komputer kalian tidak bisa koneksi internet, atau saat membuka browser diminta credentials username dan password proxy, maka buka

<https://ditsti.itb.ac.id/nic/> (lewat HP atau perangkat lain)



The image shows a blue web interface titled "Pengelolaan Akun INA" with a logo of Institut Teknologi Sepuluh Nopember (ITS) at the top left. Below the title, there are four buttons: "Pengecekan User", "Informasi Password Proxy", "Pengubahan Password", and "Reset Password". A red box highlights the "Informasi Password Proxy" button, and a callout window shows the details of this form.

Informasi Password Proxy

Username	[Redacted]
Password Proxy	[Redacted]

©2022, Direktorat Sistem dan Teknologi Informasi ITB

Praktikum Luring: Environment

Dibebaskan menggunakan IDE/Editor apapun selama membantu kalian untuk mempelajari OOP. *In other word, cara shortcut seperti github copilot, ~~ChatGPT~~, dan alat bantu lain yang membuat kalian tidak mempelajari OOP dilarang.*

Q: Kok ga ada VSCode di komputer lab, kak? :(

A: *sok install, pasang intellisense juga boleh. Yang dilarang: github copilot, ~~ChatGPT~~, dan sejenisnya.*

Note: kalau tidak bisa install, lapor ke asistennya.

Praktikum Luring: Environment

Biasakan penggunaan terminal, ini termasuk:

1. Kompilasi
2. Navigasi folder
3. Etc

* kalian sangat disarankan untuk mencoba ini di laptop/PC masing-masing setelah tutorial

Jadi idealnya, kalian...

1. Membuat kode di editor apapun
2. Compile dan run program dengan command berikut di terminal

```
$ g++ abc -o main.cpp
```

atau

```
$ g++ abc -o main.cpp Class1.cpp Class2.cpp ...
```

```
$ ./abc
```

atau

```
$ main
```

github.com/littlemight/pbo-tut-stack

Header

- Dalam C++, kita bisa menggunakan ekstensi *.h maupun *.hpp. Ada beberapa perbedaan, tapi tidak perlu dipikirkan dalam konteks praktikum ini. Defaultnya, selalu pilih hpp.
- Dalam definisi kelas, mungkin saja nama argumen tidak dispesifikasikan, misal

void push(int)

artinya, nama variabel tersebut baru akan dituliskan di implementasi method

Contoh Class: Stack

```
// stack.h
#ifndef STACK_H
#define STACK_H

class Stack {
private:
    int size;
    int capacity;
    int* data;
public:
    Stack();           // default constructor
    Stack(int cap);    // user defined constructor
    Stack(const Stack& s); // copy constructor
    ~Stack();          // destructor

    void push(int x);  // menambahkan isi stack
    int pop();          // mengambil dan menghapus top dari stack
};

#endif
```

Contoh Class: Stack (Constructor)

```
// stack.h
#ifndef STACK_H
#define STACK_H

class Stack {
private:
    int size;
    int capacity;
    int* data;
public:
    Stack();
    Stack(int cap);
    Stack(const Stack& s);
    ~Stack();
    void push(int x);
    int pop();
};

#endif
```

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

Stack::Stack() {
    this->capacity = 10;
    this->size = 0;
    this->data = new int[this->capacity];
}

Stack::Stack(int cap) {
    this->capacity = cap;
    this->size = 0;
    this->data = new int[this->capacity];
}
```


Contoh Class: Stack (Constructor)

```
// stack.h
#ifndef STACK_H
#define STACK_H

class Stack {
private:
    int size;
    int capacity;
    int* data;
public:
    Stack();
    Stack(int cap);
    Stack(const Stack& s);
    ~Stack();
    void push(int x);
    int pop();
};

#endif
```

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

Stack::Stack(const Stack& s) {
    this->capacity = s.capacity;
    this->size = s.size;
    this->data = new int[s.capacity];
    for (int i = 0; i < this->capacity; i++) {
        this->data[i] = s.data[i];
    }
}
```

Contoh Class: Stack (Destructor)

```
// stack.h
#ifndef STACK_H
#define STACK_H

class Stack {
private:
    int size;
    int capacity;
    int* data;
public:
    Stack();
    Stack(int cap);
    Stack(const Stack& s);
    ~Stack();
    void push(int x);
    int pop();
};

#endif
```

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

Stack::~Stack() {
    delete[] this->data;
}
```

Contoh Class: Stack (Method)

```
// stack.h
#ifndef STACK_H
#define STACK_H

class Stack {
private:
    int size;
    int capacity;
    int* data;
public:
    Stack();
    Stack(int cap);
    Stack(const Stack& s);
    ~Stack();
    void push(int x);
    int pop();
};

#endif
```

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

void Stack::push(int x) {
    if (this->size < this->capacity) {
        this->data[this->size] = x;
        this->size++;
    }
}

int Stack::pop() {
    int top = 0;
    if (this->size > 0) {
        this->size--;
        top = this->data[this->size];
    }
    return top;
}
```

Contoh Class: Stack (Driver)

```
#include <iostream>
#include "stack.h"
using namespace std;

int main() {
    Stack s1;
    s1.push(2);
    s1.push(3);
    cout << s1.pop() << endl;

    Stack s2(15);
    s2.push(5);
    cout << s2.pop() << endl;

    Stack s3(s1);
    s3.push(1);
    cout << s3.pop() << endl;
    cout << s3.pop() << endl;
}
```

Output:

???

Contoh Class: Stack (Driver)

```
#include <iostream>
#include "stack.h"
using namespace std;

int main() {
    Stack s1;
    s1.push(2);
    s1.push(3);
    cout << s1.pop() << endl;

    Stack s2(15);
    s2.push(5);
    cout << s2.pop() << endl;

    Stack s3(s1);
    s3.push(1);
    cout << s3.pop() << endl;
    cout << s3.pop() << endl;
}
```

Output:

3
5
1
2

Const

Class mungkin memiliki konstan: atribut konstan, atau method yang konstan.

Atribut yang konstan artinya tidak akan berubah sepanjang hidupnya objek.

Fungsi yang konstan artinya tidak akan mengubah atribut kelas.

Const

```
// stack.h
#ifndef STACK_H
#define STACK_H

class Stack {
private:
    int size;
    int capacity;
    int* data;
public:
    Stack();
    Stack(int cap);
    Stack(const Stack& s);
    ~Stack();
    void push(int x);
    int pop();
};

#endif
```

```
// stack.h
#ifndef STACK_H
#define STACK_H

class Stack {
private:
    int size;
    const int capacity;
    int* data;
public:
    Stack();
    Stack(int cap);
    Stack(const Stack& s);
    ~Stack();
    void push(int x);
    int pop();
};

#endif
```

Const

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

Stack::Stack() : capacity(10) {
    this->size = 0;
    this->data = new int[this->capacity];
}

Stack::Stack(int cap) : capacity(cap) {
    this->size = 0;
    this->data = new int[this->capacity];
}

Stack::Stack(const Stack& s) : capacity(s.capacity) {
    this->size = s.size;
    this->data = new int[s.capacity];
    for (int i = 0; i < this->capacity; i++) {
        this->data[i] = s.data[i];
    }
}
```


Const

```
// stack.h
#ifndef STACK_H
#define STACK_H

class Stack {
private:
    int size;
    const int capacity;
    int* data;
public:
    Stack();
    Stack(int cap);
    Stack(const Stack& s);
    ~Stack();

    void push(int x);
    int pop();
    int top() const;
};

#endif
```

```
// stack.cpp

...

int Stack::top() const {
    int top = 0;
    if (this->size > 0) {
        top = this->data[this->size - 1];
    }
    return top;
}
```

Static

Kelas dapat memiliki atribut dan fungsi yang statik.

Atribut yang statik artinya memiliki nilai yang sama untuk semua objek yang hidup.

Fungsi yang statik artinya dapat dipanggil meskipun tidak diinstantiasi menjadi objek.

Static Attribute

```
// stack.h
#ifndef STACK_H
#define STACK_H

class Stack {
private:
    int size;
    const int capacity;
    int* data;
    static int n_stack;
public:
    Stack();
    Stack(int cap);
    Stack(const Stack& s);
    ~Stack();

    static int getNumOfStack();
    ...
};

#endif
```

```
// stack.cpp

int Stack::n_stack = 0;

Stack::Stack() : capacity(10) {
    ...
    n_stack += 1;
}

Stack::Stack(int cap) : capacity(cap) {
    ...
    n_stack += 1;
}

Stack::Stack(const Stack& s) : capacity(s.capacity) {
    ...
    n_stack += 1;
}
```

Static Method

```
// stack.cpp
```

```
int Stack::getNumOfStack() {  
    return n_stack;  
}
```

```
// stack_driver.cpp
```

```
#include <iostream>  
#include "stack.h"  
using namespace std;
```

```
int main() {  
    Stack s1;  
    cout << Stack::getNumOfStack() << endl;  
  
    Stack s2(15);  
    cout << Stack::getNumOfStack() << endl;  
  
    Stack s3(s1);  
    cout << Stack::getNumOfStack() << endl;  
}
```

Friend

Pada defaultnya, private member dari sebuah kelas tidak dapat diakses kecuali oleh method di objek itu sendiri.

Namun hal ini dapat diubah bila sebuah kelas / fungsi dijadikan “friend”.

Sebuah fungsi / kelas yang dijadikan friend, dapat membaca private atribut dari kelas itu, namun tidak dapat mengubahnya.

Sebuah kelas pasti friend dari kelas itu sendiri.

Friend

```
// stack_driver.cpp
...
int compareStackSize(const Stack& s1, const Stack& s2) {
    int result = 0;
    if (s1.size < s2.size) {
        result = -1;
    } else if (s1.size > s2.size) {
        result = 1;
    } else {
        result = 0;
    }
    return result;
}

int main() {
    Stack s1, s2;

    ...

    cout << compareStackSize(s1, s2) << endl;
}
```

```
// stack.h
#ifndef STACK_H
#define STACK_H

class Stack {
private:
    ...
public:
    ...

    friend int compareStackSize(
        const Stack& s1, const Stack& s2);
};

#endif
```

Operator Overloading

Pada C++, kita dapat membuat definisi dari operator seperti `=`, `+`, `*`, `<<`, dan lain-lain.

Ada 2 cara untuk mendefinisikan ini, yakni menggunakan method atau membuat fungsi yang di-”friend”.

Operator Overloading

```
// stack.h
#ifndef STACK_H
#define STACK_H

class Stack {
private:
    ...
public:
    ...

    Stack operator+(const Stack& s);
    // atau
    friend Stack operator+(const Stack& s1, const Stack& s2);

};

#endif
```


Function Overloading

Di C++, kelas dapat memiliki lebih dari 1 fungsi yang memiliki argumen yang berbeda-beda.

C++ akan otomatis memilihkan method yang sesuai dengan argumen yang diberikan.

Function Overloading

```
// stack.h
#ifndef STACK_H
#define STACK_H

class Stack {
private:
    ...
public:
    ...
    void push(int x);           // memasukkan x ke stack
    void push(int x, int num); // memasukkan x sebanyak n kali ke stack
    void push(float x);        // memasukkan floor(x) ke stack
    void push(string x);       // memasukkan nilai karakter ASCII tiap karakter
                                // di string x ke stack
};

#endif
```

Pop Quiz

```
// stack.cpp
```

```
void Stack::ubah() {  
    this->size = 1;  
    this->data[0] = 9;  
}
```

```
// stack_driver.cpp
```

```
void coba(Stack s) {  
    s.ubah();  
}  
  
int main() {  
    Stack s;  
  
    s.push(2);  
    s.push(3);  
    coba(s);  
  
    cout << s.getSize() << endl;  
    cout << s.top() << endl;  
}
```

Pop Quiz

```
// stack.cpp
```

```
void Stack::ubah() {  
    this->size = 1;  
    this->data[0] = 9;  
}
```

- stack "s" (dan semua property-nya) yang didalam fungsi "coba" merupakan objek yang berbeda dengan stack "s" dalam main,
- mengikuti copy constructor yang ada pada slide 14, maka setiap properti "disalin" mengikuti implementasi copy constructor tersebut
- perubahan terhadap "s" dalam fungsi coba tidak mempengaruhi "s" yang berada dalam main

```
// stack_driver.cpp
```

```
void coba(Stack s) {  
    s.ubah();  
}
```

```
int main() {  
    Stack s;
```

```
    s.push(2);  
    s.push(3);  
    coba(s);
```

```
    cout << s.getSize() << endl; // 2  
    cout << s.top() << endl; // 3
```

```
}
```

Pop Quiz

```
// stack.cpp
```

```
void Stack::ubah() {  
    this->size = 1;  
    this->data[0] = 9;  
}
```

Bagaimana jika tidak ada copy constructor?

```
class Stack {  
public:  
    ...  
    // Stack(const Stack& s);  
    ~Stack();  
    void push(int x);  
    int pop();  
};
```

```
// stack_driver.cpp
```

```
void coba(Stack s) {  
    s.ubah();  
}  
  
int main() {  
    Stack s;  
  
    s.push(2);  
    s.push(3);  
    coba(s);  
  
    cout << s.getSize() << endl;  
    cout << s.top() << endl;  
}
```

Pop Quiz

- stack "s" (dan semua property-nya) yang didalam fungsi "coba" merupakan objek yang berbeda dengan stack "s" dalam main,
- Namun karena tidak ada copy constructor, maka C++ akan melakukan member wise copy untuk setiap property (varA = other.varA, varB = other.varB, etc)
Basically kaya gini
// ini constructor yang bakal "dibuat" dan dipakai oleh C++ kalo kita ga bikin implementasi copy constructor
Stack::Stack(const Stack& s) {
 this->capacity = s.capacity;
 this->size = s.size;
 this->data = s.data; // ini pointer
}

// stack.cpp

```
void Stack::ubah() {  
    this->size = 1;  
    this->data[0] = 9;  
}
```

Bagaimana jika tidak ada copy constructor?

```
class Stack {  
public:  
    ...  
    // Stack(const Stack& s);  
    ~Stack();  
    void push(int x);  
    int pop();  
};
```

// stack_driver.cpp

```
void coba(Stack s) {  
    s.ubah();  
}
```

```
int main() {  
    Stack s;
```

```
    s.push(2);  
    s.push(3);  
    coba(s);
```

```
    cout << s.getSize() << endl; // 2  
    cout << s.top() << endl; // random number
```

```
}
```

Misal "s" dalam fungsi "coba" kita sebut sCoba, dan "s" dalam fungsi "main" kita sebut sMain
- Perhatikan kalo data itu adalah pointer ke array, alhasil data di sCoba pointernya menunjuk ke array yang sama dengan data di sMain
(Ini ilmu di OS, nanti kalian bakal belajar):
- kalo fungsi udah kelar eksekusi, nanti OS bakal cleanup semua memori hasil alokasi untuk variabel di fungsi tersebut (untuk kasus ini, untuk sCoba)
- dan karena pointer array di sCoba dan sMain sama, maka array tersebut ikut di cleanup sama OS
- alhasil pada saat mengakses s.top(), kita mengakses alamat di memory yang telah di clean up oleh OS
:) ga usah terlalu dipikirin sekarang kalo belum ngerti

Pop Quiz

```
// stack.h

class Stack {
private:
    ...
public:
    Stack();
    Stack(int cap);
    // Stack(const Stack& s);
    ~Stack();
    ...
}
```

```
// stack_driver.cpp

...

int main() {
    Stack s1;
    s1.push(2);
    s1.push(3);

    Stack s2(s1);

    s1.push(4);
    s2.push(5);

    cout << s1.pop() << endl;
    cout << s2.pop() << endl;

}
```

Pop Quiz

```
// stack.h

class Stack {
private:
    ...
public:
    Stack();
    Stack(int cap);
    // Stack(const Stack& s);
    ~Stack();
    ...
}
```

Notes PPT:

Top S1 : 1

Top S2 : 1

Arr : 2 3 5

5

5

```
// stack_driver.cpp
```

```
...
```

```
int main() {
    Stack s1;
    s1.push(2);
    s1.push(3);

    Stack s2(s1);

    s1.push(4);
    s2.push(5);
```

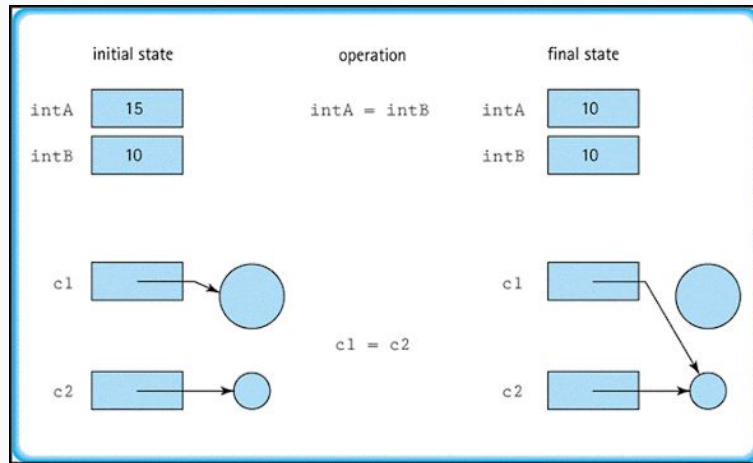
```
    cout << s1.pop() << endl; // 5
```

```
    cout << s2.pop() << endl; // 5
```

```
}
```


Untuk masa depan yang tidak bingung...

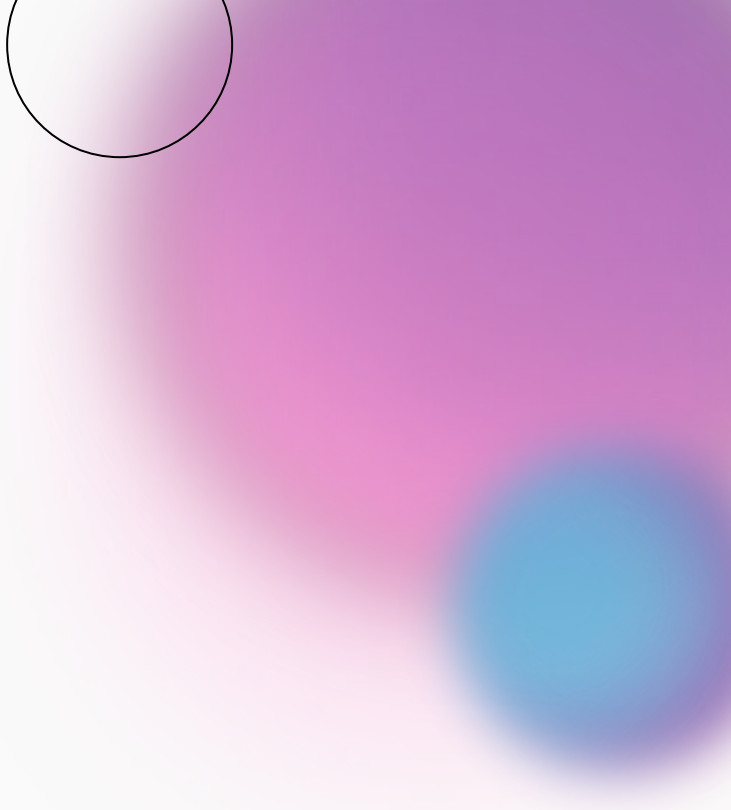
1. Melempar data harus dipertimbangkan lagi! yang dikirim itu *value/reference/pointer*???
2. Kalau mengirim bentuk data **non primitif**, itu yang dikirim *reference/pointer*, **bukan value**!



Jangan menyalahkan asisten jika jawaban praktikum tidak berjalan sesuai keinginan padahal salah konsep.

Kami menilai pemahaman kalian mengenai OOP melalui praktikum, bukan pemahaman algoritma.

Karena ini mata kuliah OOP bukan mata kuliah Dasar Pemrograman/Pengenalan komputasi.



Pertanyaan? Komentar?

Next Praktikum: tidak menerima masalah pada responsi ini. Jika bingung, tanya & selesaikan sekarang juga!



Waktunya Presensi



Sekian.

Ditunggu praktikum dan tutorial berikutnya.
