

MINGGU 1 & 2

Praktikum Pemrograman Berbasis Objek
oleh Asisten IF2210 2019/2020

PRAKTIKUM

1. Tutorial dan praktikum dilakukan bergantian
2. Saat tutorial, akan ada 30 menit berisi materi dan 70 menit mengerjakan soal
3. Saat tutorial, mahasiswa boleh bertanya pada asisten (bantu debug, dll)
4. Saat praktikum, akan ada 100 menit untuk mengerjakan soal
5. Saat praktikum, mahasiswa **tidak boleh** bertanya maupun browsing
6. Mulai minggu depan, masuk ke lab sesuai alokasi lab yang ada di olympia

OUTLINE

1. Ctor, Cctor, Dtor
2. Const
3. Friends
4. Operator Overloading
5. Function Overloading

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 (COPY 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"

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

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

    Stack s3(s1);
    s3.push(1);
    std::cout << s3.pop() << std::endl;
    std::cout << s3.pop() << std::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 DI CLASS

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.

CONTOH CONST PADA ATRIBUT (I)

```
// 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
```

CONTOH CONST PADA ATRIBUT (2)

```
// 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];
    }
}
```

CONTOH CONST PADA METHOD

```
// 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 DI CLASS

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.

CONTOH ATRIBUT STATIK DI CLASS STACK

```
// 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;
}
```

CONTOH METHOD STATIK DI CLASS STACK

```
// 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;  
}
```

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.

CONTOH FRIEND DI CLASS STACK (I)

```
// 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;
}
```

CONTOH FRIEND DI CLASS STACK (2)

```
// 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`.

CONTOH OPERATOR OVERLOADING DI CLASS STACK

```
// 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.

CONTOH FUNCTION OVERLOADING DI CLASS STACK

[illegible]

POP QUIZ: STACK

```
// 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

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

}
```

POP QUIZ: STACK

```
// 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

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

```
}
```

POP QUIZ: STACK

```
// stack.cpp
```

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

Bagaimana jika tidak ada copy constructor?

```
// 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

```
// stack.cpp
```

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

Bagaimana jika tidak ada copy constructor?

```
// 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; // 0
```

```
}
```