



**MODUL PEMROGRAMAN BERORIENTASI OBJEK  
(CCC210)**

**MODUL 07  
VARIABEL POINTER DAN MANAJEMEN MEMORI**

**DISUSUN OLEH  
INDRIANI NOOR HAPSARI, ST, MT**

Universitas  
**Esa Unggul**

**UNIVERSITAS ESA UNGGUL  
2020**

## MODUL 7 - VARIABEL POINTER DAN MANAJEMEN MEMORI

### A. Kemampuan Akhir Yang Diharapkan

Setelah mempelajari modul ini, diharapkan:

1. Mahasiswa dapat memahami variabel *pointer* dan variabel *reference*.
2. Mahasiswa memahami mekanisme manajemen memori di C++ dan Java
3. Mahasiswa dapat memahami alokasi dan dealokasi memori dengan tepat di C/C++

### B. Outline Topik

1. Manajemen Memori di C++.....	2
a) Pointer.....	2
b) Basic Memory Management di C++.....	7
c) Dynamic Array.....	8
d) Destructor.....	9
2. Memory Management di JAVA.....	10



Universitas  
**Esa Unggul**

## C. Uraian

Memory management adalah proses mengalokasikan objek baru dan menghapus objek yang tidak digunakan untuk mengembalikan ruang/memori untuk alokasi objek baru. Pada modul ini, akan dijelaskan dasar manajemen memori di C++ dan di Java.

### 1. Manajemen Memori di C++

Tempat khusus di memori disimpan untuk variabel dinamik, yang disebut dengan **freestore** atau **the heap**. Jika program Anda membuat terlalu banyak variabel dinamik, hal ini akan menghabiskan memori di **freestore**. Jika semua memori di **freestore** habis terpakai, maka pembuatan variabel dinamik berikutnya akan gagal.

Dalam c++, penggunaan dan penghapusan kembali variabel dinamik dikelola sepenuhnya oleh pemrogram. Sehingga, pemrogram harus selalu membersihkan kembali variabel dinamik apabila sudah tidak lagi digunakan. Apabila variabel dinamik dihapus, maka memori yang digunakan oleh variabel tersebut dapat digunakan kembali.

#### a) Pointer

Sebuah Pointer adalah alamat memori dari sebuah variabel. Alamat yang digunakan untuk memberi nama sebuah variabel dengan cara ini disebut dengan pointer, karena alamat tersebut dapat dianggap “menunjuk” ke variabel. Alamat ini “points”/menunjuk variabel karena mengidentifikasi variabel dengan memberitahu dimana variabel berada, ketimbang memberitahu apa nama variabel nya.

Sebagai contoh, sebuah variabel yang berada di nomor lokasi 1007 dapat ditunjuk dengan berkata “variabel yang ada di sana di lokasi 1007”. Ketika sebuah variabel digunakan sebagai argumen *call-by-reference*, alamatnya akan diberikan ke fungsi yang dipanggil (bukan nama variabel nya).

#### Variabel Pointer

Sebuah variabel yang memiliki pointer harus dideklarasikan untuk memiliki tipe pointer, dengan menempatkan tanda bintang sebelum nama variabel. Contoh:

```
double *p;
```

Operator & dapat digunakan untuk mendapatkan alamat variabel, dan kita dapat kemudian meng-*assign* alamat itu ke variabel pointer. Contoh:

```
double v1, *p1;  
p1 = &v1;
```

### Dereferencing

Setelah dilakukan assignment alamat variabel v1 ke pointer p1, maka kita bisa mengacu variabel v1 dengan dua cara:

1. Memanggil v1 langsung
2. Memanggil “variabel yang ditunjuk oleh p1” atau \*p1.

Penggunaan asteris (\*) pada \*p1 disebut sebagai operator *dereferencing*, dan variabel pointer disebut variabel yang di-dereferensi.

Sebagai contoh, perhatikan kode program berikut ini

```
v1 = 0;  
p1 = &v1;  
*p1 = 42;  
  
cout << v1 << endl;  
cout << *p1 << endl;
```

Karena v1 dan \*p1 menunjuk ke alamat yang sama, maka nilai v1 dan \*p1 sama. Dengan demikian, output dari kode program di atas adalah

```
42  
42
```

Penjelasannya adalah sebagai berikut.

Setelah dilakukan assignment berikut ini

```
p1 = &v1;
```

Maka p1 menunjuk ke variabel v1. Sehingga, \*p1 dan v1 mengacu ke variabel yang sama.

```
*p1 = 42;
```

Assignment pada kode di atas mengubah nilai `*p1`, yang juga merupakan `v1`, menjadi 42, meskipun tidak dinyatakan secara eksplisit *assignment* ke variabel `v1`.

### Operator “new”

Operator ***new*** dapat digunakan untuk membuat variabel tanpa nama (tanpa identifier). Variabel tanpa nama ini diacu melalui pointer. Variabel yang dibuat menggunakan operator ***new*** disebut dengan ***dynamic variables***, sebab, variabel tersebut dibuat dan dihapus ketika program berjalan.

Sebagai contoh, perhatikan kode program berikut ini.

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

int main()
{
    int *p1, *p2;

    p1 = new int;
    *p1 = 42;
    p2 = p1;
    cout << "*p1 = " << *p1 << endl;
    cout << "*p2 = " << *p2 << endl;

    *p2 = 53;
    cout << "*p1 = " << *p1 << endl;
    cout << "*p2 = " << *p2 << endl << endl;

    p1 = new int;
    *p1 = 88;
    cout << "*p1 = " << *p1 << endl;
    cout << "*p2 = " << *p2 << endl << endl;

    return 0;
}
```

Kode program di atas akan menghasilkan output berikut ini

```
*p1 = 42
```

```
*p2 = 42
```

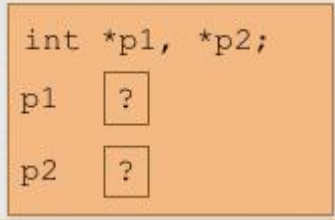
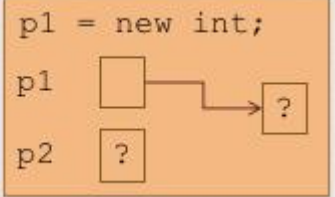
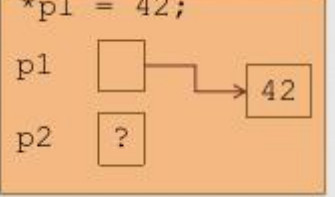
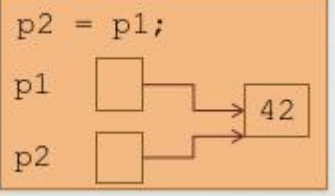
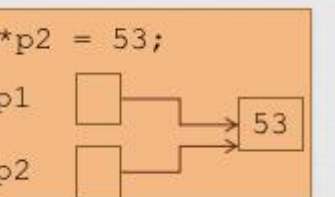
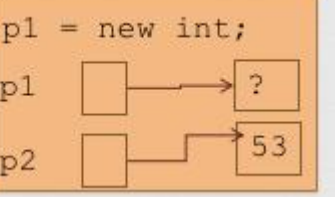
```
*p1 = 53
```

```
*p2 = 53
```

```
*p1 = 88
```

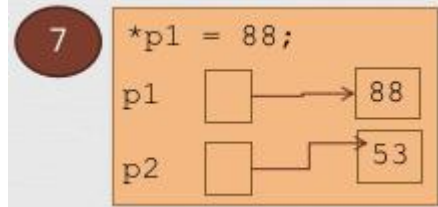
```
*p2 = 53
```

Berikut adalah penjelasan dari setiap baris kode program

<pre><b>int *p1, *p2;</b></pre> <p>Dideklarasikan 2 variabel pointer p1 dan p2</p>	<p>1</p> 
<pre><b>p1 = new int;</b></pre> <p>Variabel pointer p1 menunjuk ke variabel tanpa nama</p>	<p>2</p> 
<pre><b>*p1 = 42;</b></pre> <p>Variabel p1 di-dereferensi dan di-assign dengan nilai 42</p>	<p>3</p> 
<pre><b>p2 = p1;</b></pre> <p>Variabel pointer p1 di assign ke p2, dengan ini maka p1 dan p2 mengacu ke variabel yang sama</p>	<p>4</p> 
<pre><b>*p2 = 53;</b></pre> <p>Variabel p2 di-dereferensi dan di-assign dengan nilai 53. Karena p1 dan p2 menunjuk ke variabel yang sama, maka nilai p1 juga berubah.</p>	<p>5</p> 
<pre><b>p1 = new int;</b></pre> <p>Variabel p1 menunjuk ke variabel tanpa nama yang lain, sedangkan variabel p2 masih menunjuk ke variabel tanpa nama yang sama.</p>	<p>6</p> 

**\*p1 = 88;**

Karena p1 dan p2 telah menunjuk ke variabel yang berbeda, maka setelah assignment di atas, nilai p1 akan berubah menjadi 88, sedangkan nilai p2 tetap sama (yaitu bernilai 53)



## b) Basic Memory Management di C++

Tempat khusus di memori disimpan untuk variabel dinamik, yang disebut dengan **freestore** atau **the heap**. Jika program Anda membuat terlalu banyak variabel dinamik, hal ini akan menghabiskan memori di **freestore**. Jika semua memori di **freestore** habis terpakai, maka pembuatan variabel dinamik berikutnya akan gagal.

Dalam c++, penggunaan dan penghapusan kembali variabel dinamik dikelola sepenuhnya oleh pemrogram. Sehingga, pemrogram harus selalu membersihkan kembali variabel dinamik apabila sudah tidak lagi digunakan. Apabila variabel dinamik dihapus, maka memori yang digunakan oleh variabel tersebut dapat digunakan kembali.

Untuk menghapus variabel dinamik, digunakan operator **delete**. Operator **delete** menghapuskan variabel dinamik dan mengembalikan memori yang sebelumnya ditempati oleh variabel dinamik tersebut ke **freestore** sehingga memori tersebut dapat digunakan kembali.

Kode program berikut ini akan menghapus variabel dinamik yang ditunjuk oleh variabel p, dan mengembalikan memori yang digunakan oleh variabel dinamik tersebut ke **freestore**.

```
delete p;
```



### c) Dynamic Array

Dynamic array adalah array yang ukurannya tidak didefinisikan saat kita menulis program, melainkan ditentukan ketika program berjalan. Variabel array juga merupakan variabel *pointer*, yang menunjuk ke elemen pertama dari variabel array. Dengan deklarasi dua variabel berikut ini, **p dan a sama-sama merupakan variabel pointer** yang menunjuk ke elemen pertama dari *array*.

```
int a[10];  
int *p;
```

Oleh karenanya, nilai dari a dapat di-assign ke variabel pointer p sebagai berikut:

```
p = a;
```

Namun, *assignment* sebaliknya tidak bisa dilakukan. Kita tidak bisa mengubah nilai pointer di variabel array seperti ini

```
a = p //ILLEGAL
```

### Kapan menggunakan Dynamic Array?

Array statik harus didefinisikan ukurannya ketika kita menulis program, namun kita mungkin tidak tahu ukuran array yang diperlukan sampai program dijalankan. Untuk menghindari persoalan ini, maka dapat digunakan **dynamic arrays**. Dynamic arrays dibuat dengan menggunakan operator **new**.

```
double *a;  
a = new double[10];
```

Untuk mengembalikan memori yang digunakan oleh dynamic array ke freestore, perintah delete diikuti dengan sepasang kurung siku (*square brackets*) sebagai berikut:

```
delete [] a;
```

#### d) Destructor

Sebuah destruktur, hampir sama dengan konstruktor, merupakan fungsi spesial yang memiliki nama yang sama dengan nama kelas, dengan awalan `~`, misalkan, `~Circle()`. Destruktor dipanggil secara implisit ketika sebuah objek sudah tidak lagi digunakan. Jika Anda tidak mendefinisikan destruktur, maka compiler akan membuatkan *default destructor* namun tidak melakukan apapun.

Apabila Kelas yang Anda definisikan memiliki member data yang dialokasi secara dinamik (via `new` atau `new[]`, seperti contoh pada variabel *pointer* dan *dynamic array* yang telah dijelaskan sebelumnya), maka Anda perlu membersihkan memori dengan perintah **`delete`** atau **`delete[]`**. Hal ini perlu dituliskan secara eksplisit di dalam destruktur.

Contoh:

```
class myClass{
    private:
        int *p;
        double *a;

    public:
        ...
        //destructor
        ~myClass(){
            delete p;
            delete [] a;
        }
};
```

## 2. Memory Management di JAVA

Seperti dalam C++, objek Java yang dibuat dengan operator *new* juga akan menggunakan sebuah tempat yang disebut ***the heap***. Bedanya, pada Java proses pembersihan memori dilakukan secara otomatis oleh “***garbage collector***”, sehingga pemrogram tidak perlu menanganinya sendiri secara langsung.

***The heap*** dibuat ketika JVM (Java Virtual Machine) berjalan dan ukurannya dapat bertambah atau berkurang selama program berjalan. Ketika ***the heap*** penuh, maka ***garbage*** dikumpulkan. Selama proses ***garbage collection***, objek yang tidak lagi digunakan akan dibersihkan, sehingga tempatnya dapat digunakan kembali oleh objek baru.

The heap terkadang dibagi menjadi dua bagian, yang disebut dengan ***nursery*** dan ***old space***.

Nursery merupakan bagian dari the heap yang disimpan untuk mengalokasi objek baru. Ketika nursery penuh, *garbage is collected* dengan menjalankan “*young collection*”, dimana semua objek yang sudah lama tidak digunakan dipindahkan ke “*old space*”, sehingga *nursery* kosong kembali dan dapat digunakan untuk alokasi objek yang baru. Ketika *old space* penuh, *garbage is collected* di *old space*. Proses ini disebut dengan “*old collection*”.

#### D. Latihan

1. *Memory management* adalah proses mengalokasikan objek baru dan menghapus objek yang tidak digunakan untuk mengembalikan ruang/memori untuk alokasi objek baru.
  - a) Benar
  - b) Salah
  - c) Tidak diketahui
2. *Pointer* adalah alamat memori dari sebuah variabel.
  - a) Benar
  - b) Salah
  - c) Tidak diketahui
3. Sebuah variabel yang memiliki *pointer* dideklarasikan dengan menempatkan tanda bintang sebelum nama variabel.
  - a) Benar
  - b) Salah
  - c) Tidak diketahui
4. Ukuran array statik dapat berubah saat program berjalan.
  - a) Benar
  - b) Salah
  - c) Tidak diketahui
5. Array statik harus didefinisikan ukurannya ketika kita menulis program.
  - a) Benar
  - b) Salah
  - c) Tidak diketahui
6. Array dinamik adalah array yang ukurannya tidak didefinisikan saat kita menulis program, melainkan ditentukan ketika program berjalan.
  - a) Benar
  - b) Salah
  - c) Tidak diketahui
7. Tempat khusus di memori disimpan untuk variabel dinamik, yang disebut dengan ***freestore*** atau ***the heap***.
  - a) Benar
  - b) Salah

- c) Tidak diketahui
8. Jika program Anda membuat terlalu banyak variabel dinamik, hal ini akan menghabiskan memori di **freestore**.
- a) Benar  
b) Salah  
c) Tidak diketahui
9. Jika semua memori di **freestore** habis terpakai, maka pembuatan variabel dinamik berikutnya akan gagal.
- a) Benar  
b) Salah  
c) Tidak diketahui
10. pada Java proses pembersihan memori dilakukan secara otomatis oleh "**garbage collector**", sehingga pemrogram tidak perlu menanganinya sendiri secara langsung.
- a) Benar  
b) Salah  
c) Tidak diketahui
11. Dalam C++, penggunaan dan penghapusan kembali variabel dinamik dikelola sepenuhnya oleh pemrogram. Sehingga, pemrogram harus selalu membersihkan kembali variabel dinamik apabila sudah tidak lagi digunakan.
- a) Benar  
b) Salah  
c) Tidak diketahui
12. Untuk membaca sebuah teks file, diperlukan array statik yang ukurannya sudah didefinisikan saat menulis program.
- a) Benar  
b) Salah  
c) Tidak diketahui
13. Untuk membaca sebuah teks file, diperlukan array dinamik yang ukurannya didefinisikan saat program berjalan.
- a) Benar  
b) Salah  
c) Tidak diketahui

14. Pada C++, untuk mengembalikan memori yang digunakan oleh dynamic array ke freestore, perintah delete diikuti dengan sepasang kurung siku (*square brackets*) sebagai berikut: delete [] a.
- a) Benar
  - b) Salah
  - c) Tidak diketahui



## E. Daftar Referensi

Walter Savitch, *Problem Solving with C++*, Pearson International Edition, 2006.

[http://www.mu.ac.in/myweb\\_test/MCA study material/](http://www.mu.ac.in/myweb_test/MCA_study_material/)

<http://www.cs.fsu.edu/~xyuan/cop3330/>

<https://www.programiz.com/cpp-programming>

[buildingjavaprograms.com](http://buildingjavaprograms.com)

[https://docs.oracle.com/cd/E13150\\_01/jrockit\\_jvm/jrockit/geninfo/diagnos/garbage\\_collect.html](https://docs.oracle.com/cd/E13150_01/jrockit_jvm/jrockit/geninfo/diagnos/garbage_collect.html)

