FAKULTAS
TEKNOLOGI
INFORMASI
Universitas Advent Indonesia

Lecture 04 – Memory Management

Friday, September 22, 2023

Data Structure

fti.unai.edu

Memory Management

- Alokasi memori adalah proses menyisihkan bagian memori dalam program yang akan digunakan untuk menyimpan variabel, contoh struktur dan kelas.

- Ada 2 cara untuk melakukan alokasi memori untuk penyimpanan data:
  1. **Static** Allocation of Memory
  2. **Dynamic** Allocation of Memory

Static Allocation of Memory

Dikenal juga dengan *compile time allocation* dimana memori untuk variabel-variabel dialokasikan oleh *compiler*. Ukuran dan penyimpanan yang tepat harus diketahui pada waktu kompilasi dan untuk deklarasi array, ukurannya harus konstan.

Contoh:

int x, y; → compiler akan mengalokasikan 4 byte untuk setiap variabel x dan y.

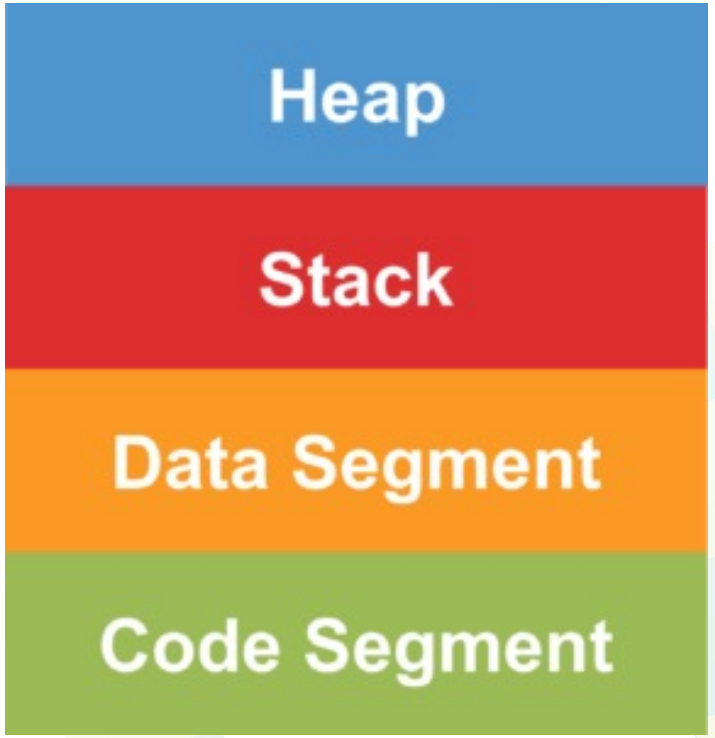
float a[5]; → compiler akan mengalokasikan 20 byte untuk array a (5*4, karena terdapat 5 elemen dan setiap elemen berukuran 4 byte)

Static Allocation of Memory

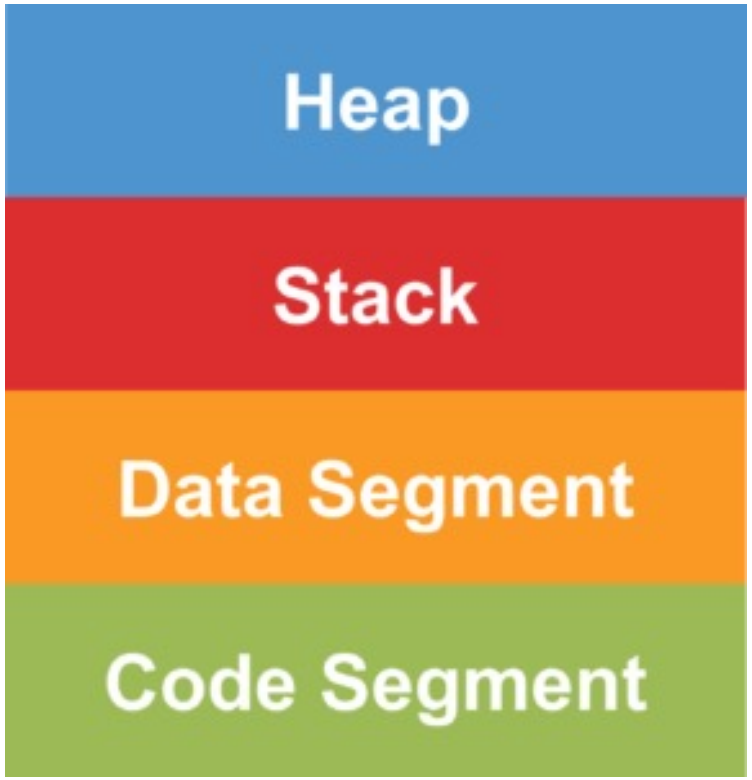
- *Arrray* biasanya digunakan untuk menyimpan beberapa data data homogen.
- Ketika *arrray* pertama kali dideklarasikan, kita harus mengalokasikan memori *arrray* dimana umumnya kebutuhan akan memori tidak dapat ditentukan sampai *runtime*.
- Hal terbaik yang dapat dilakukan untuk mengatasi situasi tersebut adalah dengan mendeklarasikan *arrray* dengan ukuran maksimum memori yang memungkinkan. Namun, hal ini menyebabkan memori yang tidak terpakai tidak dapat digunakan oleh program yang lain.
- Solusi: **dynamic memory allocation** (alokasi memori dinamis)

Dynamic Allocation of Memory

- Dikenal juga dengan *runtime allocation* dimana memori dialokasikan pada saat runtime dan alokasi ruang memori dilakukan secara dinamis selama program dijalankan. Dalam hal ini, ukuran ruang atau jumlah data item yang tepat tidak harus diketahui oleh compiler. Pointer memainkan peranan penting disini.
- Memori yang dialokasikan biasanya terdapat pada Heap.



Memory Architecture of C++ Programs



1. **Code Segment**: berisi semua kode program yang akan dieksekusi. Sifatnya adalah *read only*.
2. **Data Segment**: berisi variabel global dan variabel statis. Sifatnya tidak *read only* oleh karena nilai dari variabel dapat diubah saat *runtime*.
3. **Stack**: Biasanya merupakan pre-allocated memory. Setiap variabel baru akan dimasukkan ke dalam stack dan ketika variabel dikeluarkan dari stack maka memori untuk variabel tersebut akan dilepaskan sehingga dapat digunakan oleh variabel lain. Ukuran stack dapat berubah setiap saat. Stack menyimpan data/variabel lokal, alamat, argument yang dikirimkan dari fungsi, dan status saat ini (current) dari memory.
4. **Heap**: Memori dialokasikan selama eksekusi program. Memori dialokasikan menggunakan operator **new** dan mendealokasi memori menggunakan operator **delete**.

Dynamic Allocation of Memory

Alokasi dinamis memerlukan dua kriteria:

1. Menciptakan ruang dinamis dalam memori
2. Menyimpan alamatnya dalam sebuah pointer (sehingga ruang dapat diakses)

Dealokasi memori juga bagian dari konsep ini dimana proses “pembersihan” ruang memori dilakukan untuk penyimpanan variable atau data lainnya.

Static vs. Dynamic Allocation of Memory

Perbedaan utama antara alokasi memori statis dan dinamis adalah:

Static Memory Allocation	Dynamic Memory Allocation
In this case, variables get allocated permanently	In this case, variables get allocated only if your program unit gets active
Allocation is done before program execution	Allocation is done during program execution
It uses the data structure called stack for implementing static allocation	It uses the data structure called heap for implementing dynamic allocation
Less efficient	More efficient
There is no memory reusability	There is memory reusability and memory can be freed when not required

“New” Operator

- C++ mendefinisikan dua operator unary yakni **new** dan **delete** untuk mengalokasikan dan mendealokasi memori selama runtime. Pointer memberikan dukungan yang diperlukan untuk sistem alokasi memori dinamis di C++.
- Operator **new** menunjukkan permintaan untuk alokasi memori dinamis pada Heap. Jika memori yang cukup tersedia maka operator **new** menginisialisasi memori dan mengembalikan alamat memori yang baru dialokasikan dan diinisialisasi ke variabel pointer.

“New” Operator

- **Syntax:**

```
datatype *pointer_variable = new datatype;
OR
datatype *pointer_variable;
pointer_variable = new datatype;
OR
pointer-variable = new data-type[size];
```

“Delete” Operator

- Setelah memori Heap dialokasikan ke objek variabel atau kelas menggunakan operator **new**, kita dapat membatalkan alokasi ruang memori menggunakan operator **delete**.
- **Syntax**:

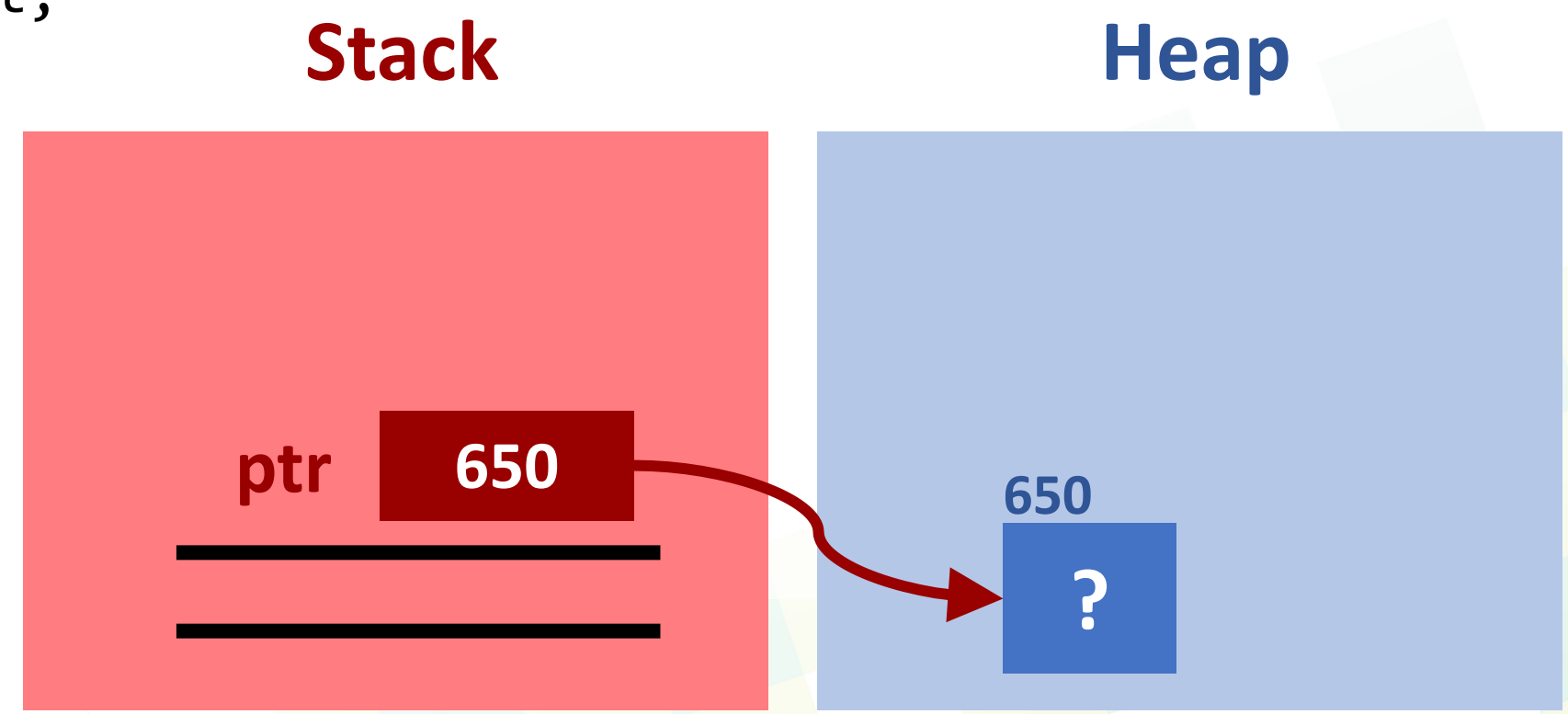
```
delete pointer_variable;

delete[] pointer_variable;
```

"New" Operator

Contoh:

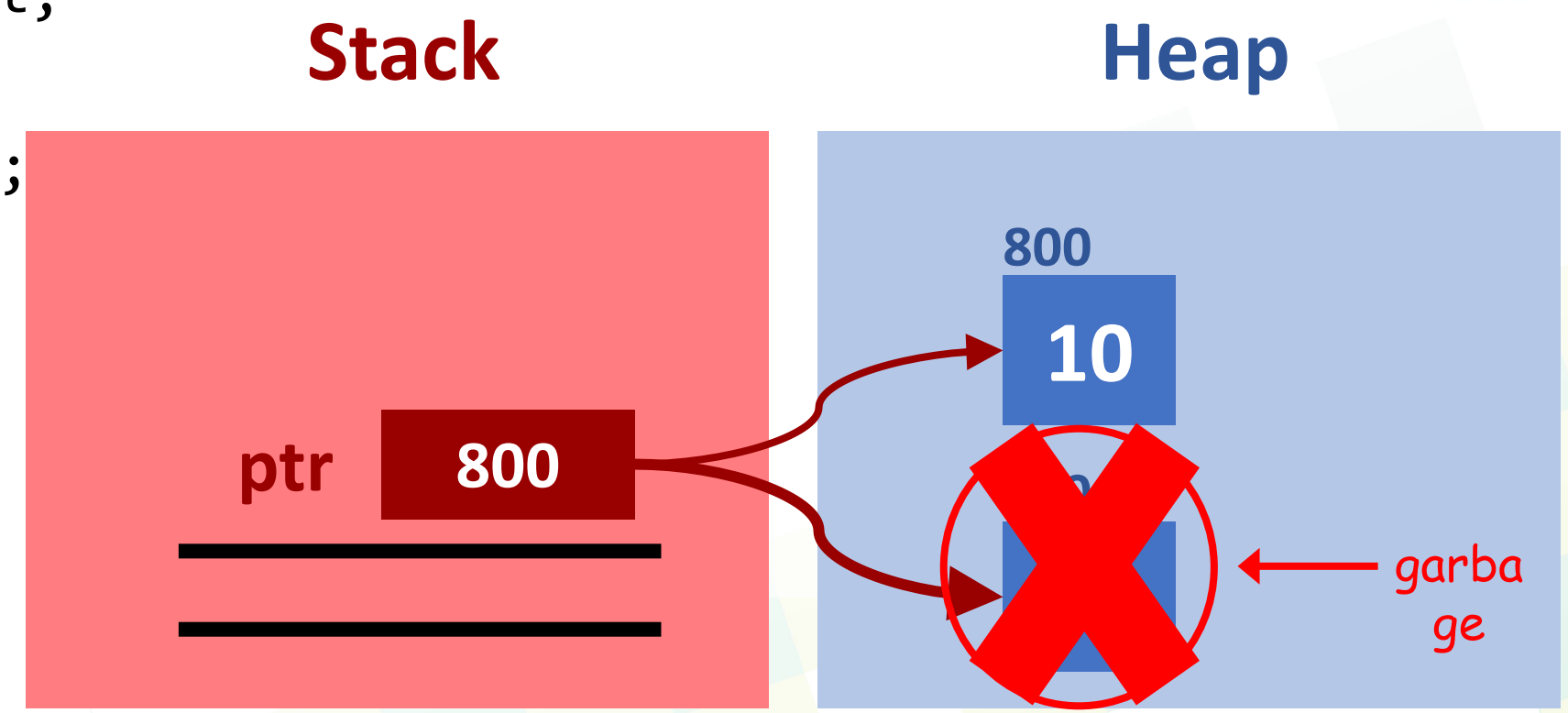
```
new int;
int *ptr = new int;
```



"New" Operator

Contoh:

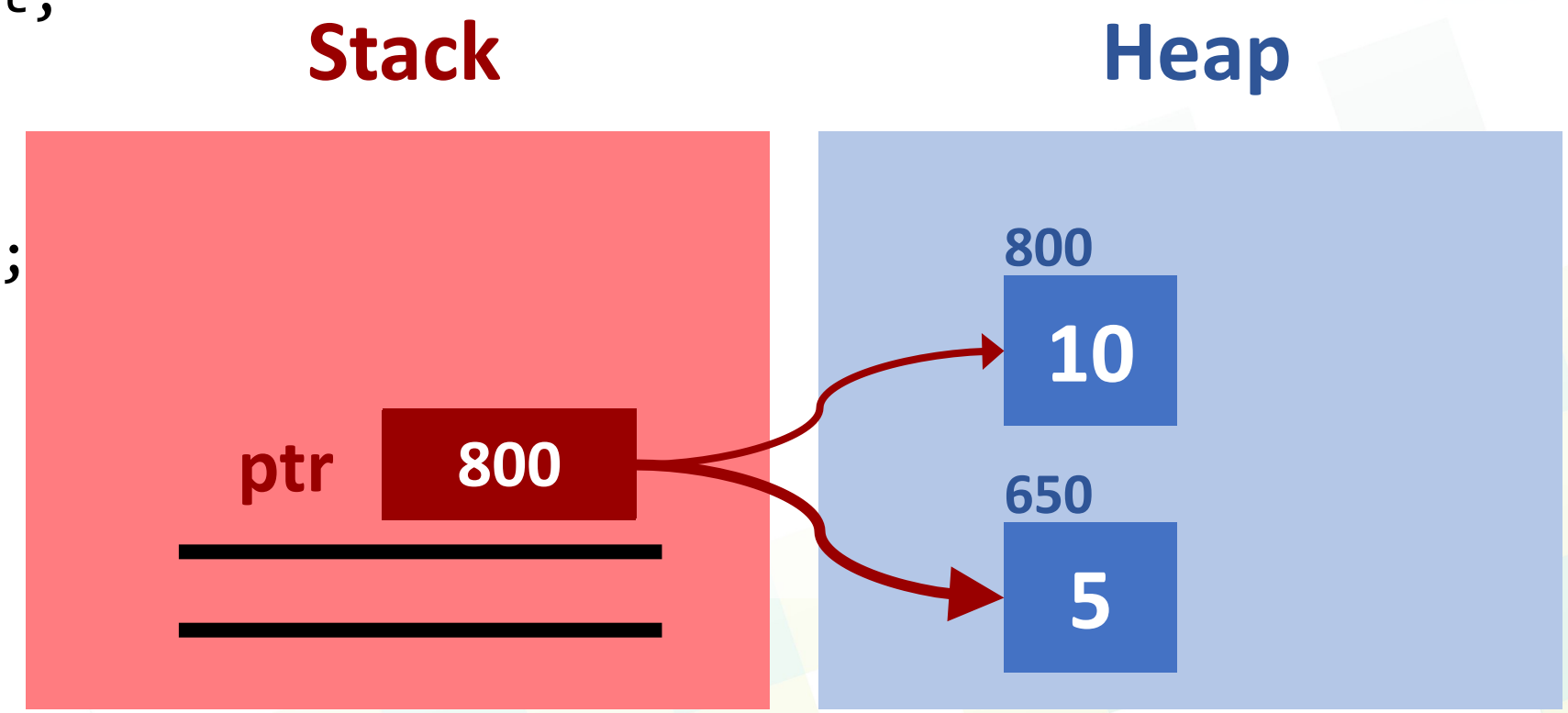
```
1 new int;
2 int *ptr = new int;
3 *ptr = 5;
4 ptr = new int(10);
```



"Delete" Operator

Contoh:

```
new int;
int *ptr = new int;
*ptr = 5;
delete ptr;
ptr = new int(10);
```



"Delete" Operator

Contoh:

```
new int;
int *ptr = new int;
*ptr = 5;
delete ptr;
```

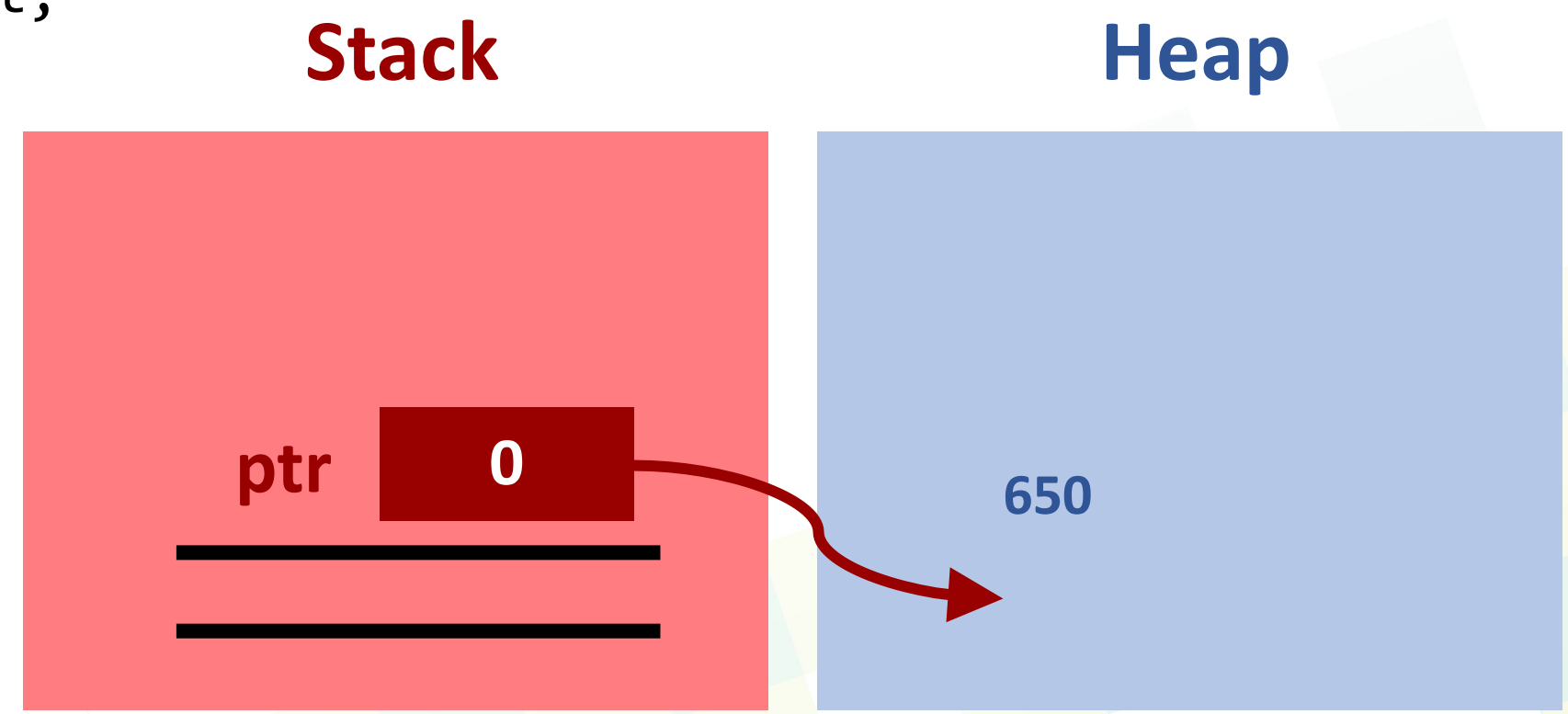
Solusi:

```
ptr = NULL;
```

atau

```
ptr = 0;
```

Dangling pointer: kondisi dimana pointer menunjuk ke lokasi memori dari objek yang sudah dihapus.



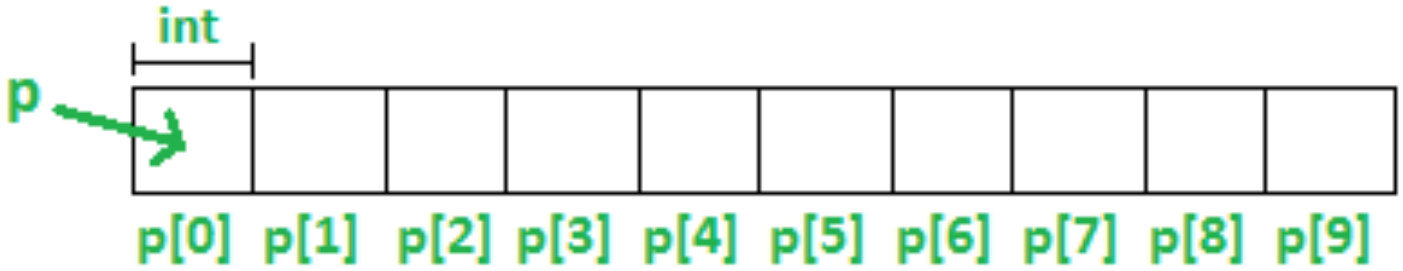
Allocate block of memory

- Operator new dapat juga digunakan untuk mengalokasikan satu blok (array) memori dengan tipe data tertentu.
- Syntax:

```
pointer-variable = new data-type[size];
```

Contoh:

```
int *p = new int[10];
```



Allocate block of memory

*Normal Array Declaration vs Using **New** Operator*

- Perbedaan yang paling signifikan adalah proses dealokasi pada *normal array* akan dilakukan oleh *compiler* (ketika fungsi selesai dijalankan), sedangkan *array* yang dialokasikan secara dinamis menggunakan operator ***new*** akan selalu ada di memori sampai didealokasikan oleh *programmer* atau program yang dijalankan berakhir.

Memory Not Available

Bagaimana jika memori tidak tersedia selama *runtime*?

- Jika memori heap yang ada tidak tersedia untuk dialokasikan, maka permintaan alokasi yang baru tidak akan terjadi atau akan diindikasikan kegagalan dengan mengirimkan suatu pengecualian (throwing an exception) dengan tipe ***std::bad_alloc***.
- Untuk itu, ketika akan mengalokasikan memori secara dinamis, perlu menggunakan kata kunci (*keyword*) "**nothrow**" bersamaan dengan operator **new**.