

# C++ For C Coders 5

**Data Structures**  
**C++ for C Coders**

한동대학교 김영섭 교수  
idebtor@gmail.com

dynamic memory allocation  
new & delete operators

# Three kinds of memory (or data)

- **Static memory**
  - where global and static variables live
  - allocated **at compile time**
- **Heap memory**
  - dynamically allocated **at execution time**
  - "managed" memory accessed using pointers
  - explicitly allocated and deallocated using operators **new** and **delete** by programmer
- **Stack memory**
  - used by automatic variables
  - automatically created at function entry, resides in activation frame of the function, and **is destroyed when returning from function**

## **Static Memory**

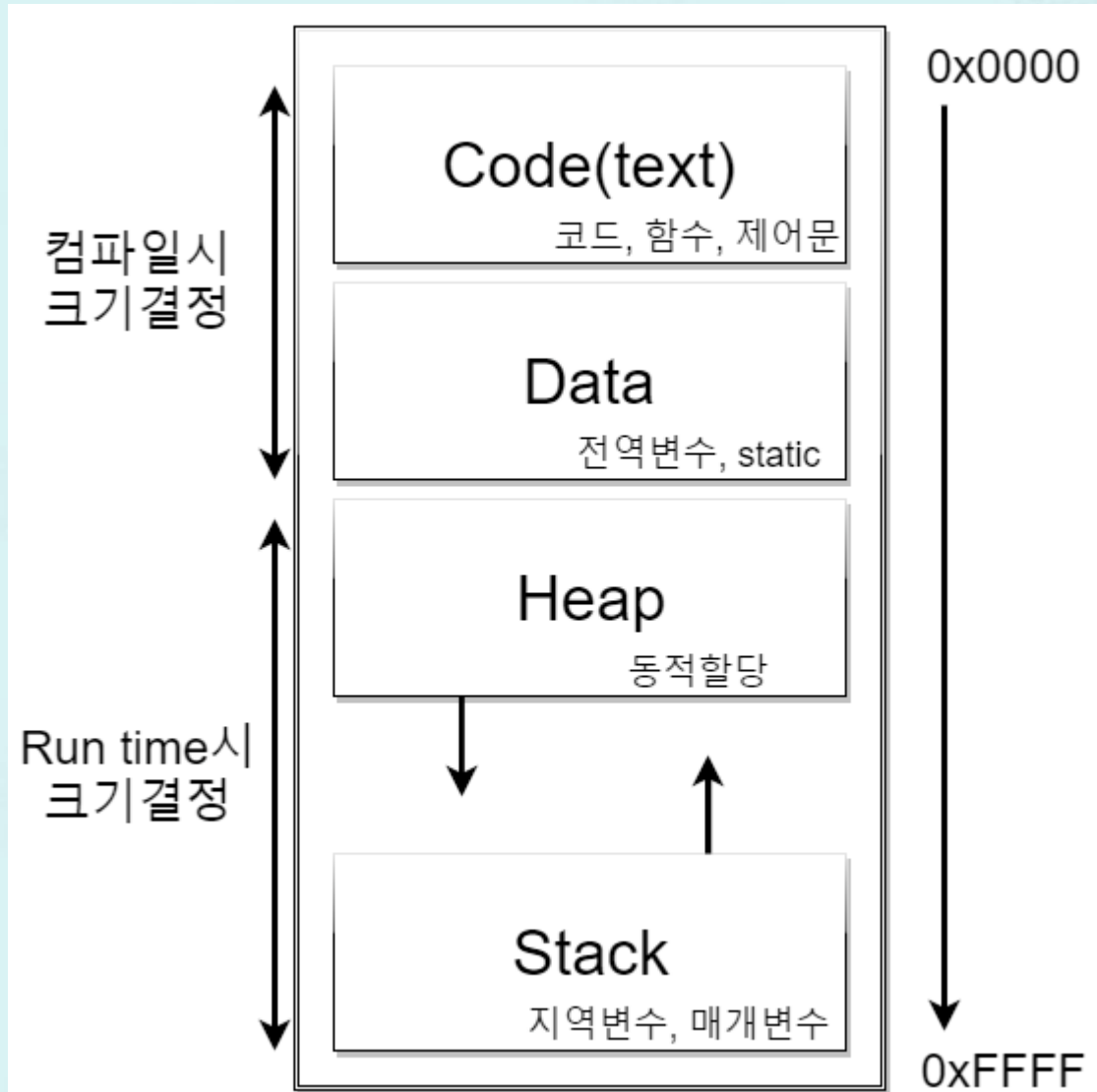
Global Variables  
Static Variables

**Heap Memory** (or free store)  
Dynamically Allocated Memory  
(Unnamed variables)

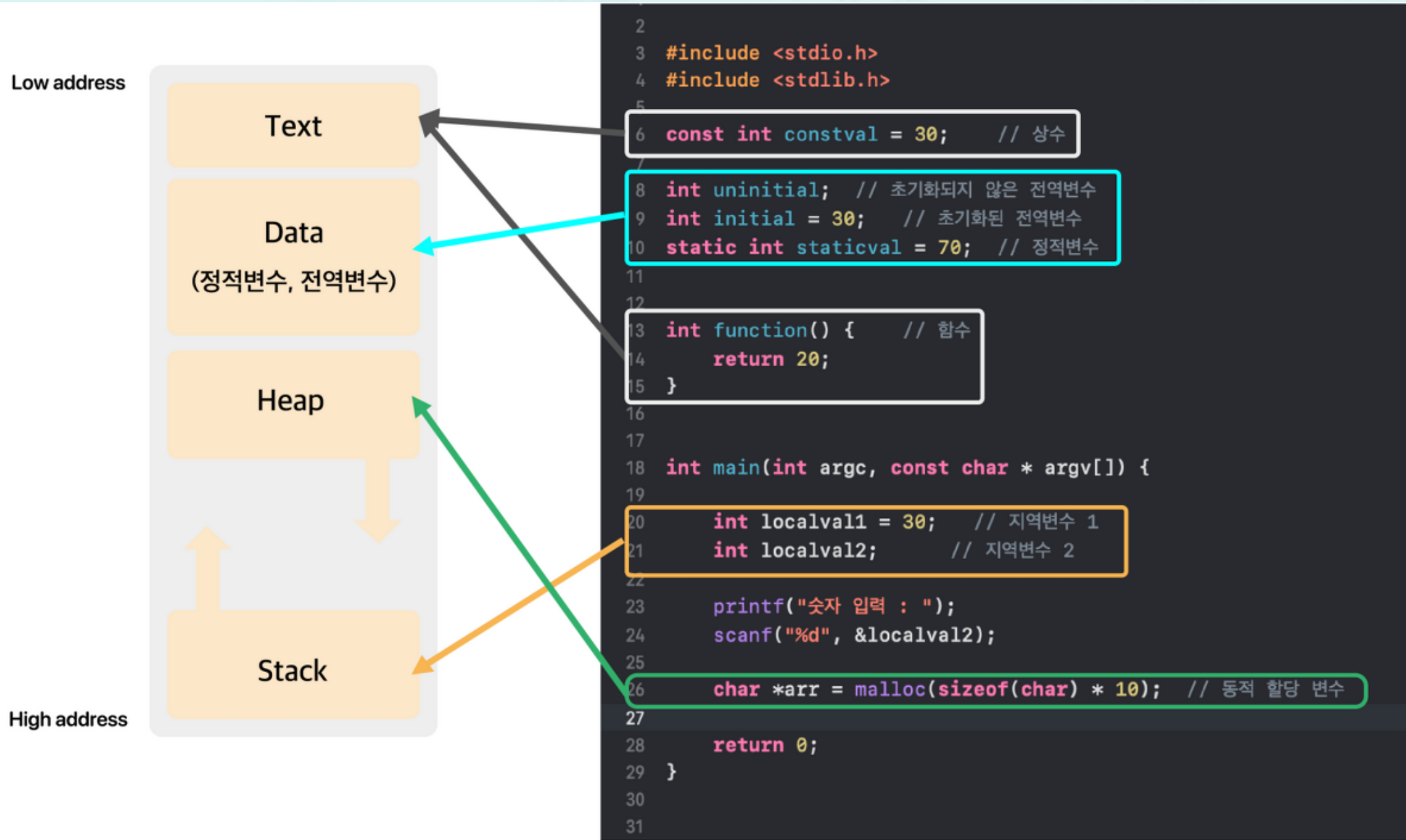
## **Stack Memory**

Auto Variables  
Function parameters

# Dynamic Memory Allocation Diagram



# Dynamic Memory Allocation Diagram



## Dynamic Memory Allocation

---

- *In C*, functions such as **malloc()** are used to dynamically allocate memory from the **Heap**.
- *In C++*, this is accomplished using the **new** and **delete** operators
- **new** is used to allocate memory during execution time
  - returns a pointer to the address where the object is to be stored
  - always returns a pointer to the type that follows the **new**

## Operator new Syntax

---

**new DataType**

**new DataType[IntialExpression]**

- If memory is available, in an area called the heap (or free store) **new** allocates the requested object or array, and returns a pointer to (address of ) the memory allocated.
- Otherwise, program terminates with error message.
- The dynamically allocated object exists until the **delete** operator destroys it.

## Operator **new**



```
char *ptr;
```

```
ptr = new char;
```

```
*ptr = 'B';
```

```
cout << *ptr;
```

**2000**

**ptr ?**



**NOTE:** Dynamic data has no variable name

## Operator **new**

```
char *ptr;
```

➔ `ptr = new char;`

```
*ptr = 'B';
```

```
cout << *ptr;
```



**NOTE:** Dynamic data has no variable name



## Operator **new**

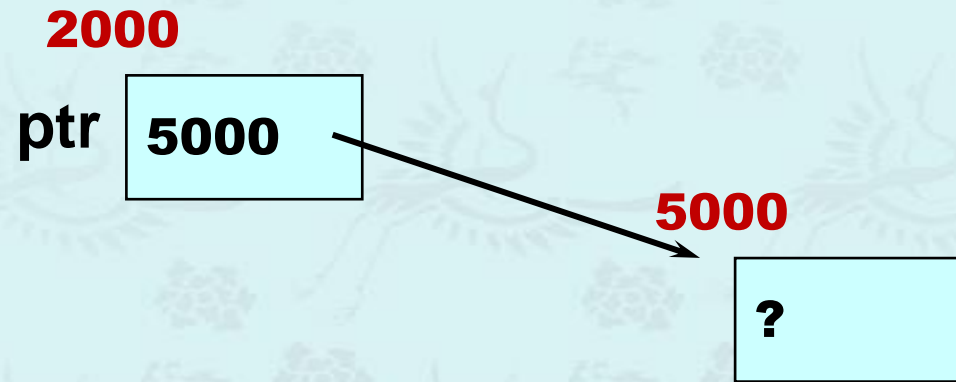
```
char *ptr;
```



```
ptr = new char;
```

```
*ptr = 'B';
```

```
cout << *ptr;
```



**NOTE:** Dynamic data has no variable name

## Operator **new**

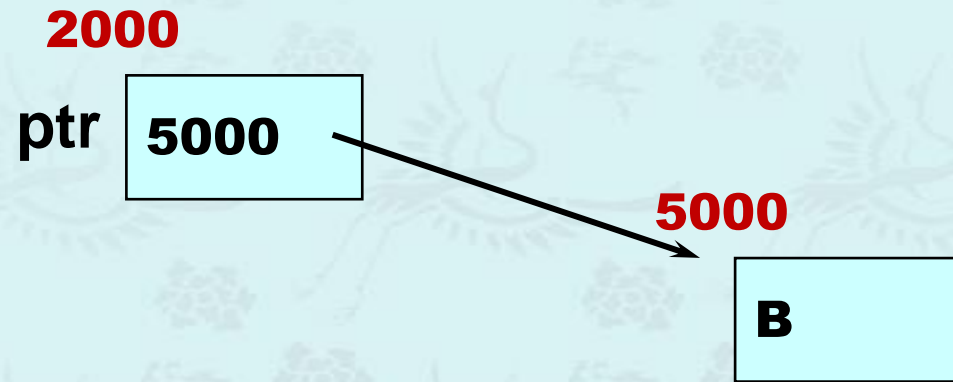
```
char *ptr;
```

```
ptr = new char;
```



```
*ptr = 'B';
```

```
cout << *ptr;
```



**NOTE:** Dynamic data has no variable name

## new vs. malloc()

---

- **new** is an operator.
- It calls the constructor.
- It returns exact data type if memory is available.
- It throws `bad_alloc` exception on failure. Use **nothrow** for **nullptr**.
- It can be overridden.
- In which memory allocated from the heap.
- Size is calculated by the compiler.
- **malloc** is a library function.
- It does not call the constructor.
- It returns the `void *` if memory is available.
- It returns **nullptr** on failure.
- It cannot be overridden.
- In which memory allocated from the heap.
- Need to pass the size.

NOTE: Use **malloc()** only if asked. Use **new** and **delete** operators in this course.

# Dynamic Memory Allocation

---

- *In C*, functions such as **malloc()** and **free()** are used to dynamically allocate and deallocate memory from the **Heap**.
- *In C++*, this is accomplished using the **new** and **delete** operators
- **new** is used to allocate memory during execution time
  - returns a pointer to the address where the object is to be stored
  - always returns a pointer to the type that follows the **new**

## The NULL/nullptr Pointer

- There is a pointer constant called the “null pointer” denoted by NULL/nullptr.
- NULL is int type 0 in C/C++, but nullptr is std::nullptr\_t type.
- **NOTE:** It is an error to dereference a pointer whose value is NULL or nullptr. Such an error may cause your program to crash, or behave erratically. It is the programmer's job to check for this.

```
while (ptr != nullptr) {  
    . . .  
    . . .                // ok to use ptr here  
}
```

## Operator **delete** Syntax

---

```
delete PointerVariable
```

```
delete [] PointerVariable
```

- The **object or array currently pointed to by Pointer is deallocated**, and the value of Pointer is undefined. The memory is returned to the free store.
- Good idea to set the pointer to the released memory to nullptr.
- Square brackets are used with delete to deallocate a dynamically allocated array.

## Operator delete

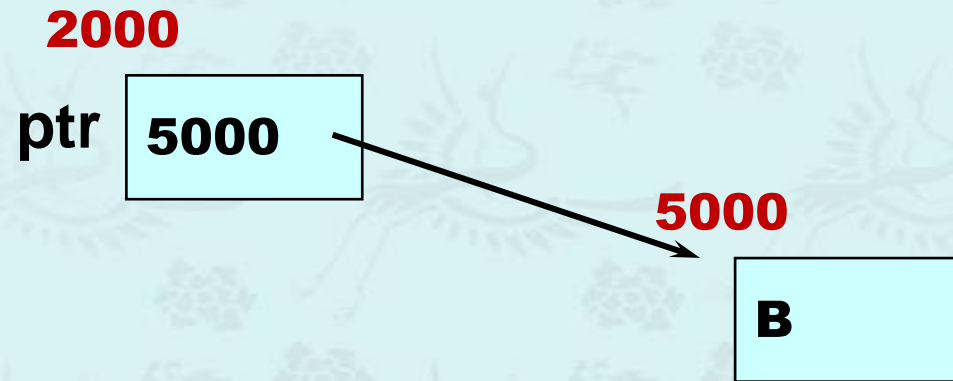
```
char *ptr;
```

```
ptr = new char;
```

➡ 

```
*ptr = 'B';
```

```
delete ptr;
```



**Question:** After 'delete' operation, can we use ptr again?

## Operator delete

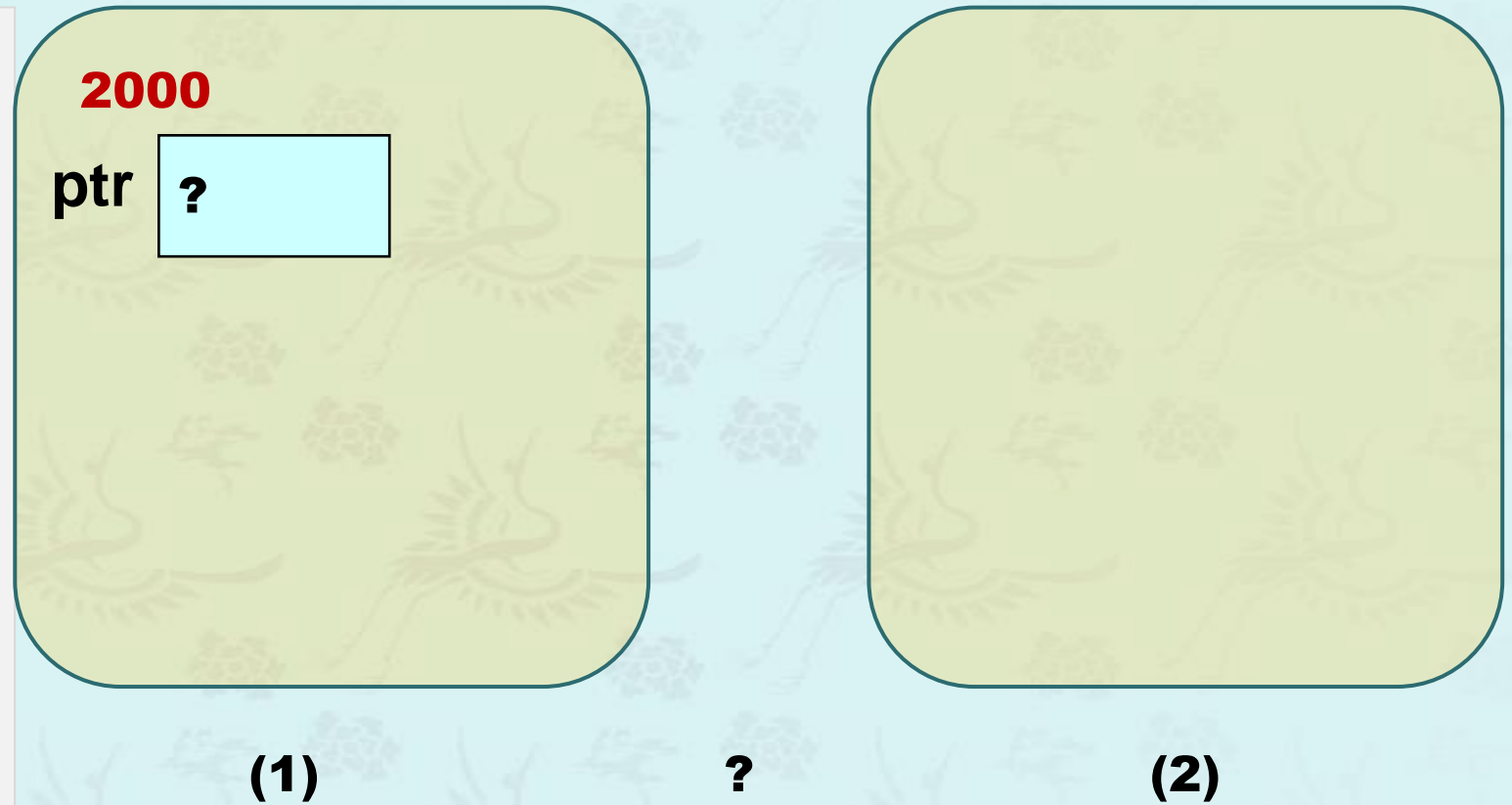
```
char *ptr;
```

```
ptr = new char;
```

```
*ptr = 'B';
```

➔ 

```
delete ptr;
```



**Question:** After 'delete' operation, can we use ptr again?



## Operator delete

```
char *ptr;
```

```
ptr = new char;
```

```
*ptr = 'B';
```

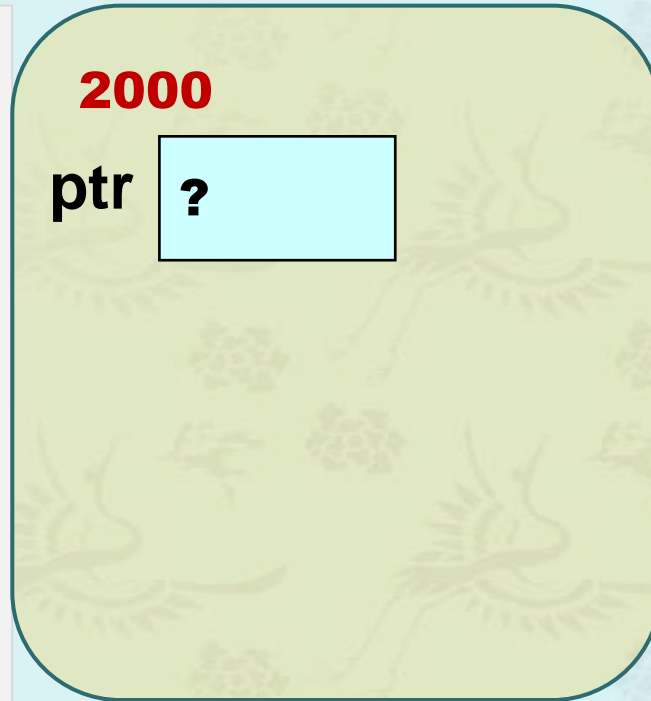
➔ 

```
delete ptr;
```

```
ptr = new char[10];
```

```
delete [] ptr;
```

**NOTE:** **delete** deallocates the memory pointed to by ptr



(1)

## Example: Operator **delete**



```
char *ptr;
```

```
ptr = new char[5];
```

```
strcpy(ptr, "Bye");
```

```
ptr[0] = 'E';
```

```
delete [] ptr;
```

```
ptr = nullptr;
```

**3000**

**ptr ?**



## Example: Operator **delete**

```
char *ptr;
```

➔ 

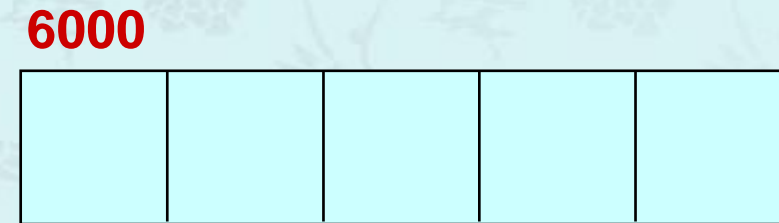
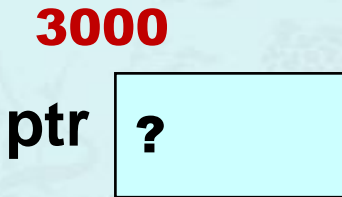
```
ptr = new char[5];
```

```
strcpy(ptr, "Bye");
```

```
ptr[0] = 'E';
```

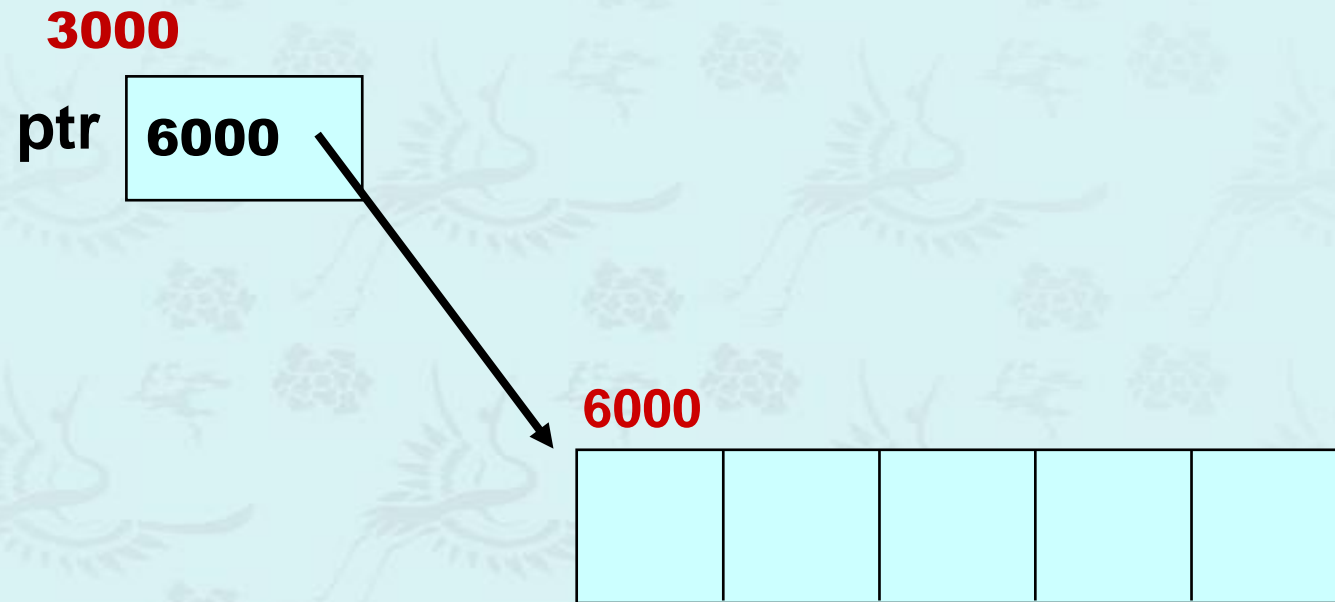
```
delete [] ptr;
```

```
ptr = nullptr;
```



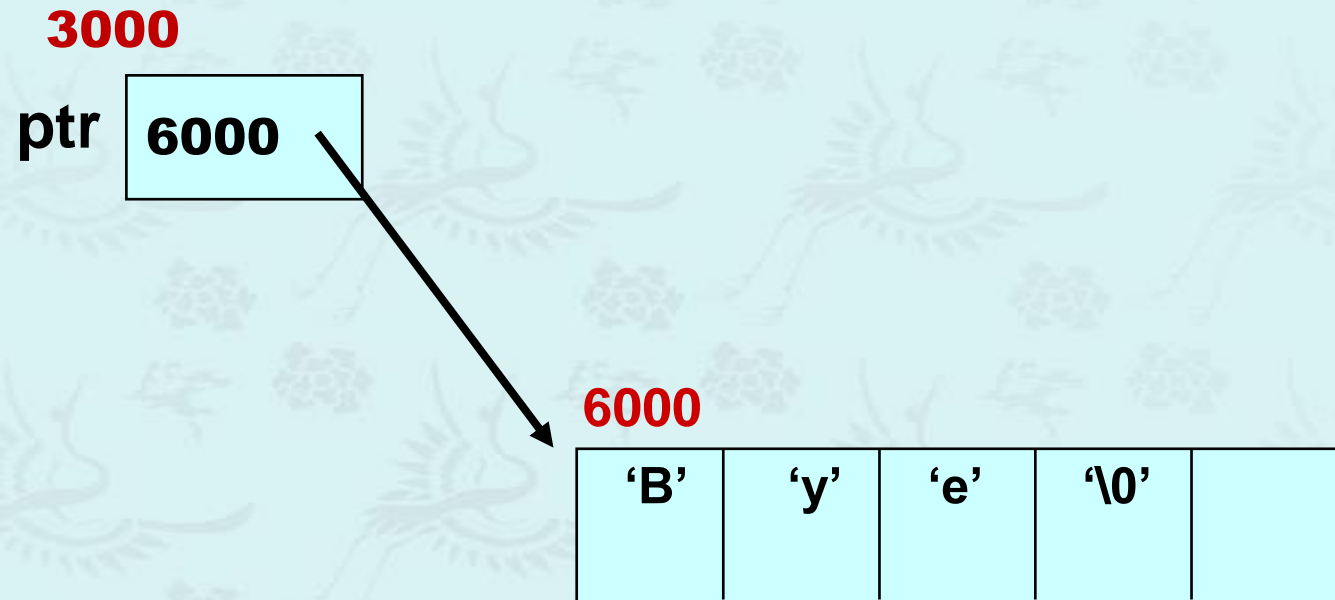
## Example: Operator **delete**

```
char *ptr;  
→ ptr = new char[5];  
strcpy(ptr, "Bye");  
ptr[0] = 'E';  
delete [] ptr;  
ptr = nullptr;
```



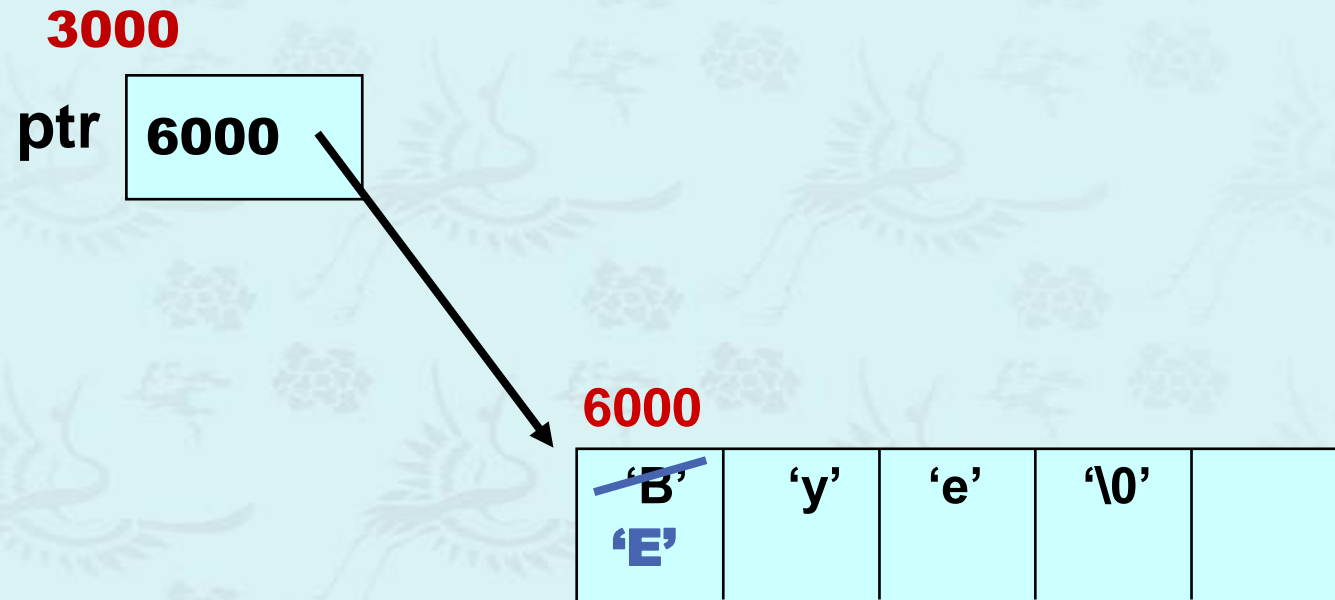
## Example: Operator **delete**

```
char *ptr;  
  
ptr = new char[5];  
  
➔ strcpy(ptr, "Bye");  
  
ptr[0] = 'E';  
  
delete [] ptr;  
  
ptr = nullptr;
```



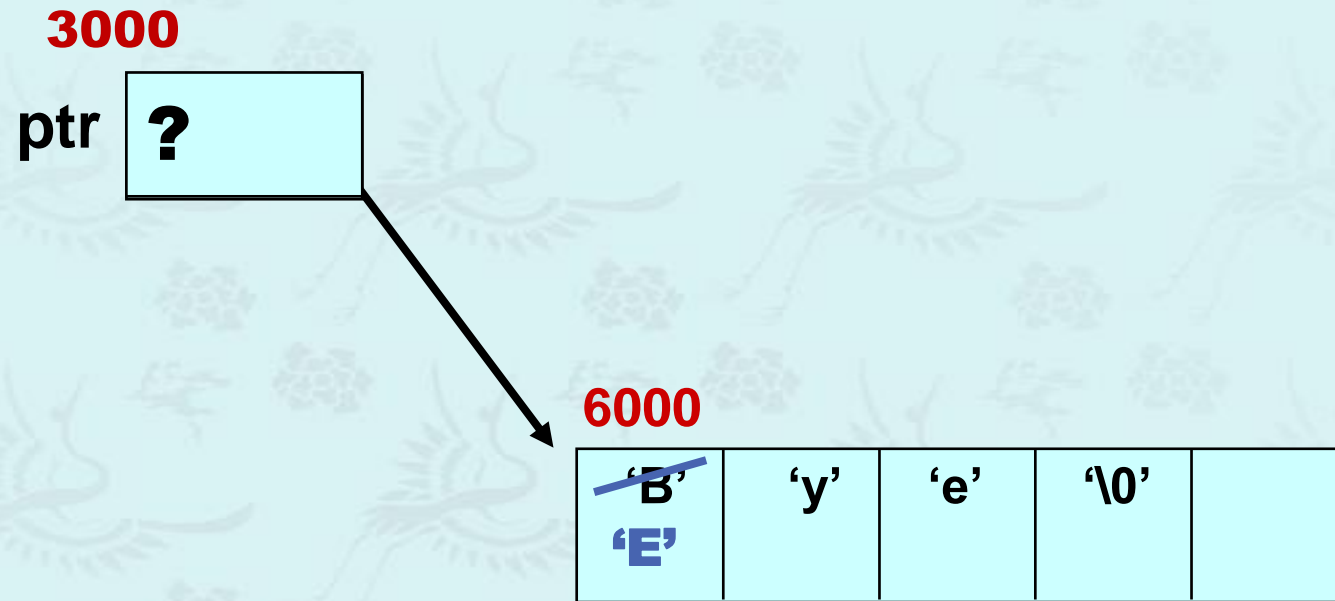
## Example: Operator **delete**

```
char *ptr;  
  
ptr = new char[5];  
  
strcpy(ptr, "Bye");  
  
→ ptr[0] = 'E';  
  
delete [] ptr;  
  
ptr = nullptr;
```



## Example: Operator **delete**

```
char *ptr;  
  
ptr = new char[5];  
  
strcpy(ptr, "Bye");  
  
ptr[0] = 'E'  
  
→ delete [] ptr;  
  
ptr = nullptr;
```



### NOTE:

- deallocates the array pointed to by ptr
- ptr itself is not deallocated
- the value of ptr becomes undefined

## Example: Operator **delete**

```
char *ptr;  
  
ptr = new char[5];  
  
strcpy(ptr, "Bye");  
  
ptr[0] = 'E'  
  
delete [] ptr;  
  
→ ptr = nullptr;
```

**3000**

**ptr NULL**

### **NOTE:**

- deallocates the array pointed to by ptr
- ptr itself is not deallocated
- the value of ptr becomes undefined



## Take Home Message

---

- Be aware of where a pointer points to, and what is the size of that space.
- Have the same information in mind when you use reference variables.
- Always check if a pointer points to nullptr before accessing it. For example,

```
char *ptr = new char[5];  
assert(ptr != nullptr);
```



## Take Home Message

---

- Be aware of where a pointer points to, and what is the size of that space.
- Have the same information in mind when you use **reference** variables.
- Always check if a pointer points to nullptr before accessing it. For example,

```
char *ptr = new char[5];  
assert(ptr != nullptr);
```



```
char *ptr = new (nothrow) char[5];  
assert(ptr != nullptr);
```

# C++ For C Coders 5

**Data Structures**  
**C++ for C Coders**

한동대학교 김영섭 교수  
idebtor@gmail.com

dynamic memory allocation  
new & delete operators