

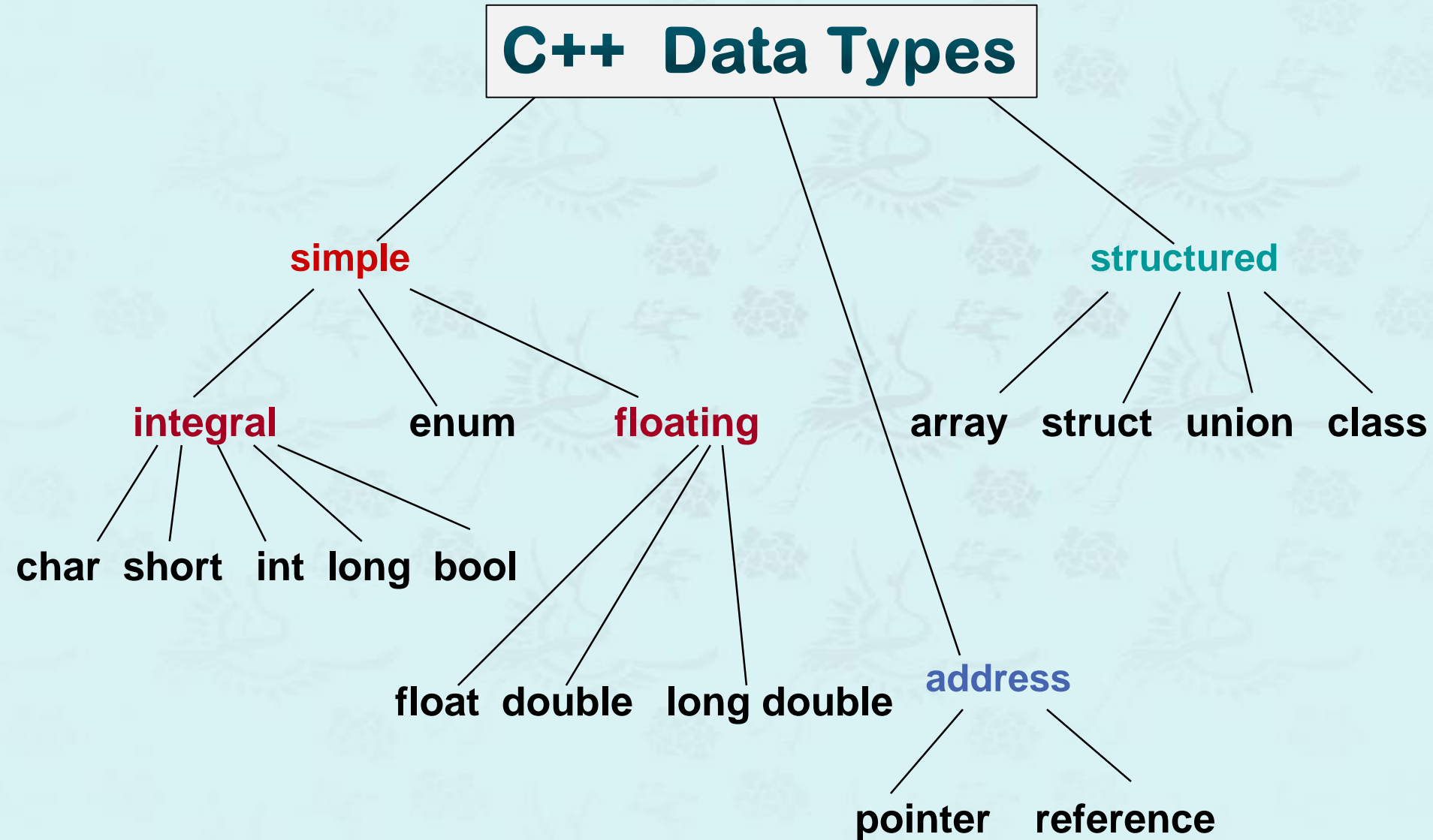
C++ For C Coders 4

Data Structures
C++ for C Coders

한동대학교 김영섭 교수
idebtor@gmail.com

pointer
address-of operator
reference & dereference operators

C++ Data Types

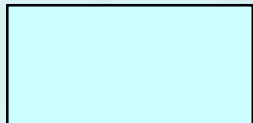


Addresses in Memory

- When a variable **is declared**, enough memory to hold a value of that type **is allocated** for it at an unused memory location.

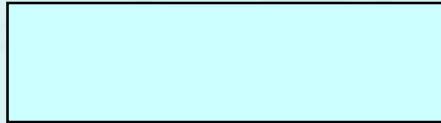
```
int      x;  
double   number;  
char     ch;
```

2000



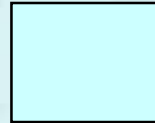
x

2004



number

2012






ch

Let's suppose that this is
the address of the variable



Obtaining Memory Addresses

- The address of a *non-array variable* can be obtained by using the **address-of operator &**
- You can **print** the address or **save** it in a variable, which is called a pointer.

int	x;	2000	2004	2012
double	number;			
char	ch;	x	number	ch


```
cout << "Address of x is " << &x << endl;
```



```
int *p = &x;
```



```
cout << "Address of x is " << ____ << endl;
```

```
x is an _____  
&x is an _____  
p is an _____  
*p is an _____
```

What is a pointer variable?

- A pointer variable is a variable whose value is **the address** of a location in memory.
- To declare a pointer variable, you must specify the type of value that the pointer will point to, for example,

```
int    *ptr; // ptr can hold the address of an int
char   *q;   // q can hold the address of a char
```

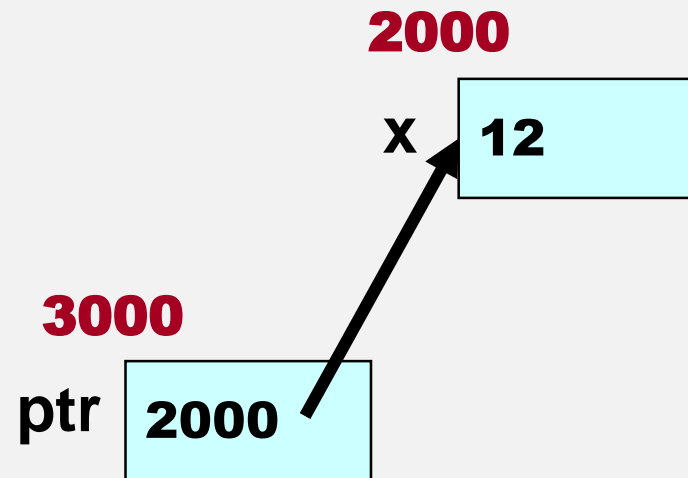
Using a Pointer Variable

```
int x;
```

```
x = 12;
```

```
int *ptr;
```

```
ptr = &x;
```



- NOTE: Because ptr holds the address of x, we say that ptr “points to” x

Using the Dereference Operator *

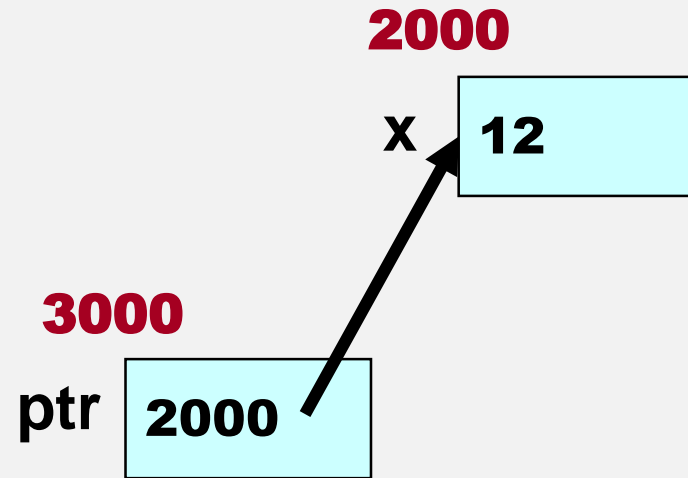
```
int x;
```

```
x = 12;
```

```
int *ptr;
```

```
ptr = &x;
```

```
cout << *ptr
```



- NOTE: The value pointed to by ptr is denoted by *ptr

Using the Dereference Operator *

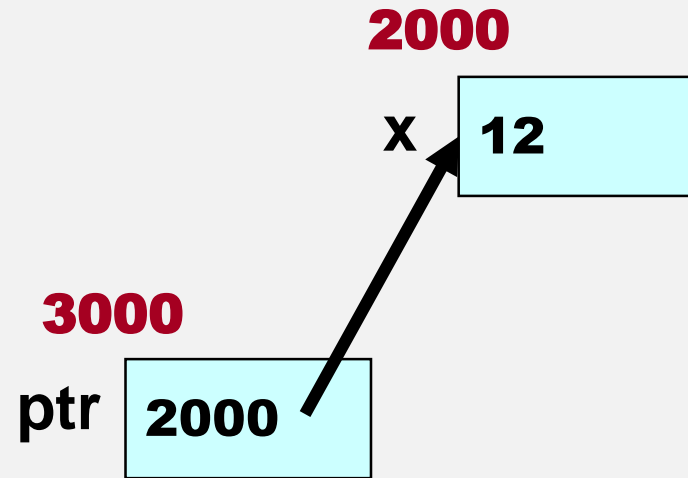
```
int x;
```

```
x = 12;
```

```
int *ptr;
```

```
ptr = &x;
```

```
*ptr = 5;
```



- NOTE: changes the value at the address ptr points to 5

Using the Dereference Operator *

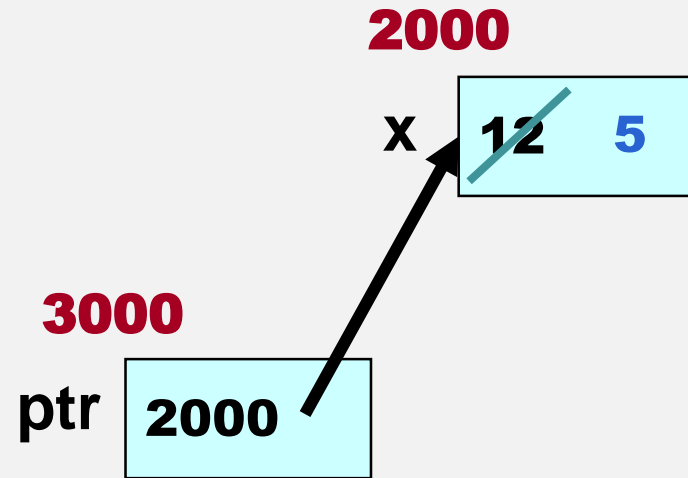
```
int x;
```

```
x = 12;
```

```
int *ptr;
```

```
ptr = &x;
```

```
*ptr = 5;
```

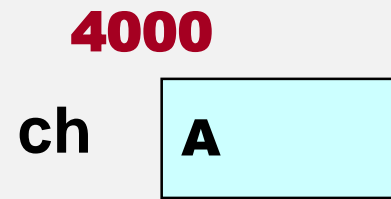


- NOTE: changes the value at the address ptr points to 5

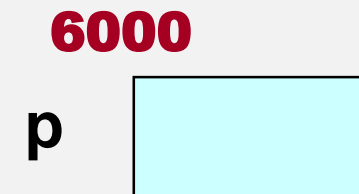
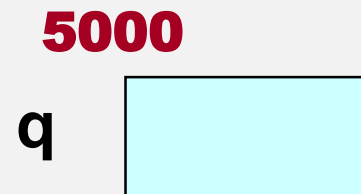
Self –Test on Pointers

Complete the diagram and fix it if necessary.

```
char ch;  
ch = 'A';
```



```
char *q;  
q = &ch;
```



```
*q = 'Z';  
char *p;
```

```
p = q;
```

Self-Test on Pointers

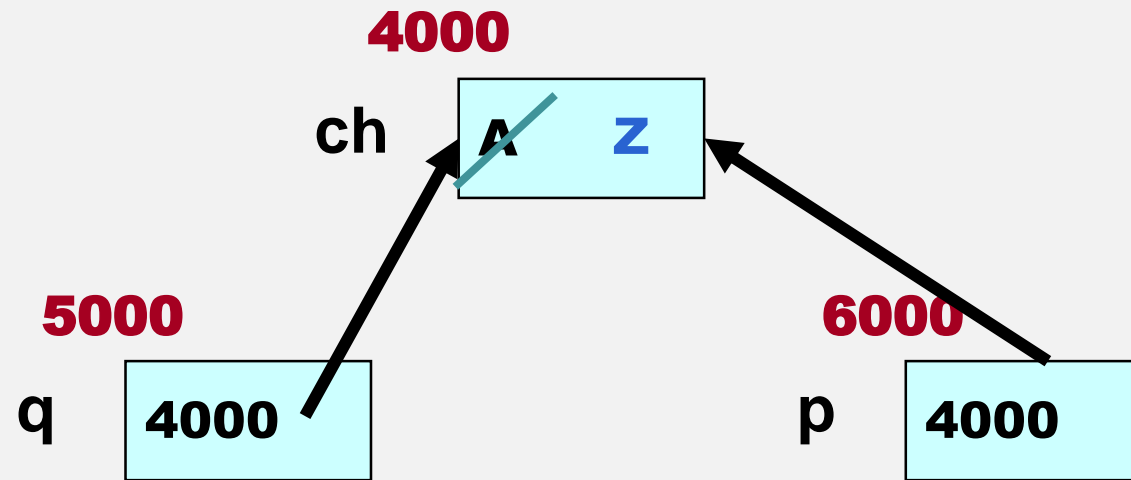
Complete the diagram and fix it if necessary.

```
char ch;  
ch = 'A';
```

```
char *q;  
q = &ch;
```

```
*q = 'Z';  
char *p;
```

```
p = q;
```

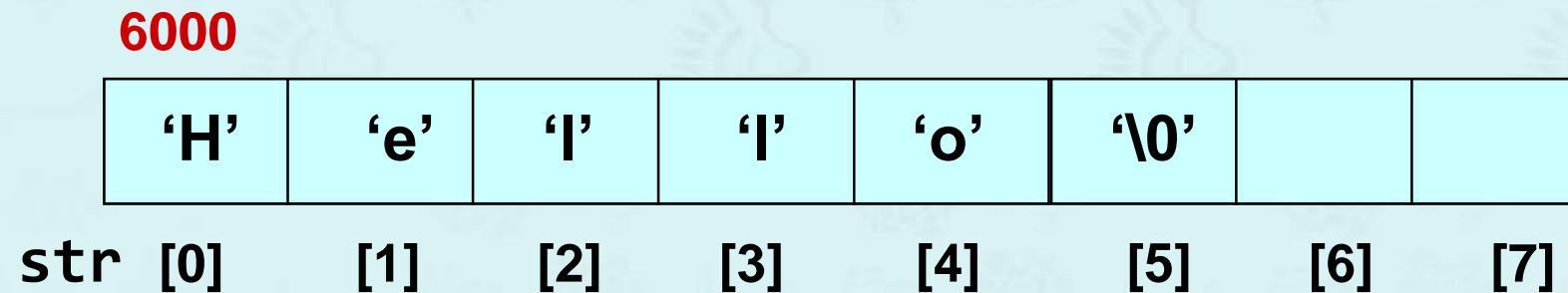


- NOTE: now `p` and `q` both point to `ch`.

Recall that . . .

```
char str[8];
```

- **str** is the base address of the array.
- We say **str** is **a pointer** because its value is an address.
- It is a pointer constant because the value of **str** itself cannot be changed by assignment. It “points” to the memory location of a char.



Using a Pointer to Access the Elements of a String

```
➔ char    msg[] = "Hello";  
char*    ptr;  
ptr      = msg;  
*ptr     = 'M' ;  
ptr++;  
  
*ptr = 'a';
```

Using a Pointer to Access the Elements of a String

➔ `char msg[] = "Hello";`

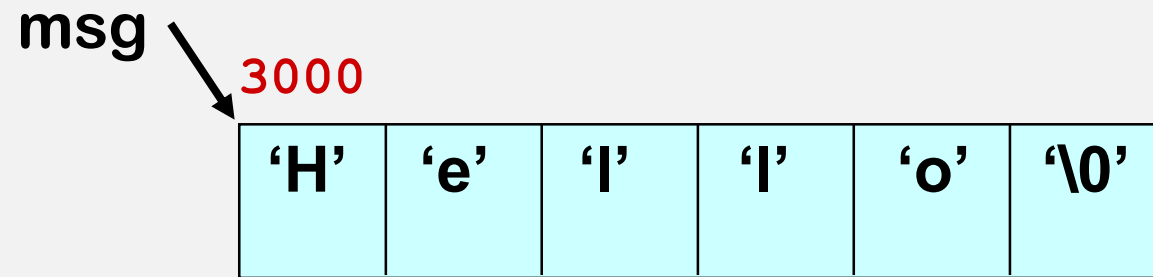
`char* ptr;`

`ptr = msg;`

`*ptr = 'M' ;`

`ptr++;`

`*ptr = 'a';`



Using a Pointer to Access the Elements of a String

```
char    msg[] = "Hello";
```

➔

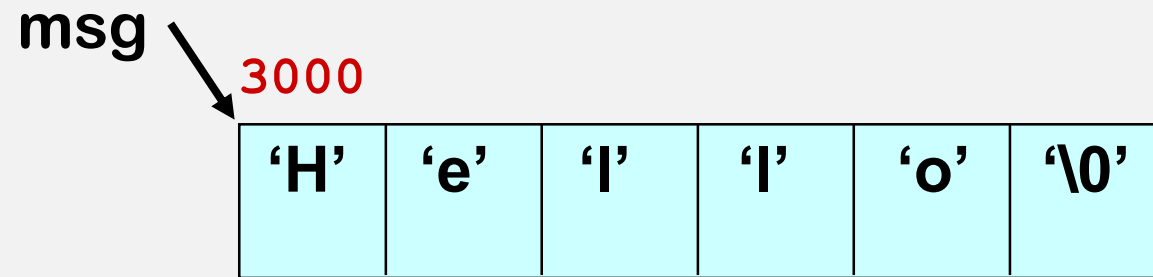
```
char*   ptr;
```

```
ptr     = msg;
```

```
*ptr    = 'M' ;
```

```
ptr++;
```

```
*ptr = 'a';
```



Using a Pointer to Access the Elements of a String

```
char    msg[] = "Hello";
```

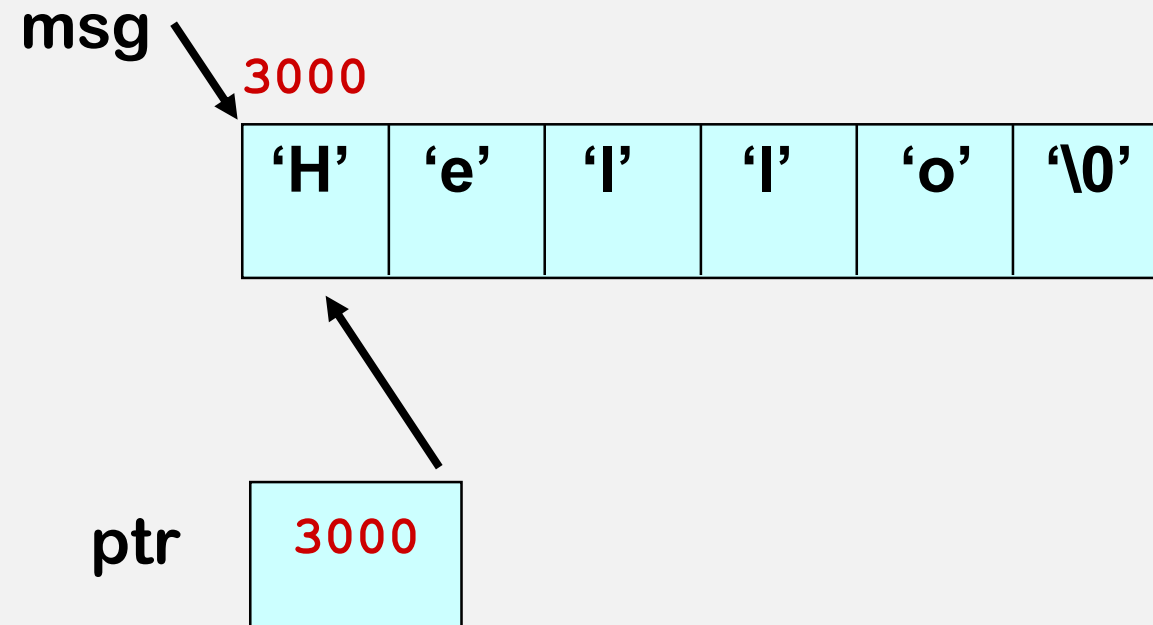
```
char*   ptr;
```

```
➔ ptr    = msg;
```

```
*ptr    = 'M' ;
```

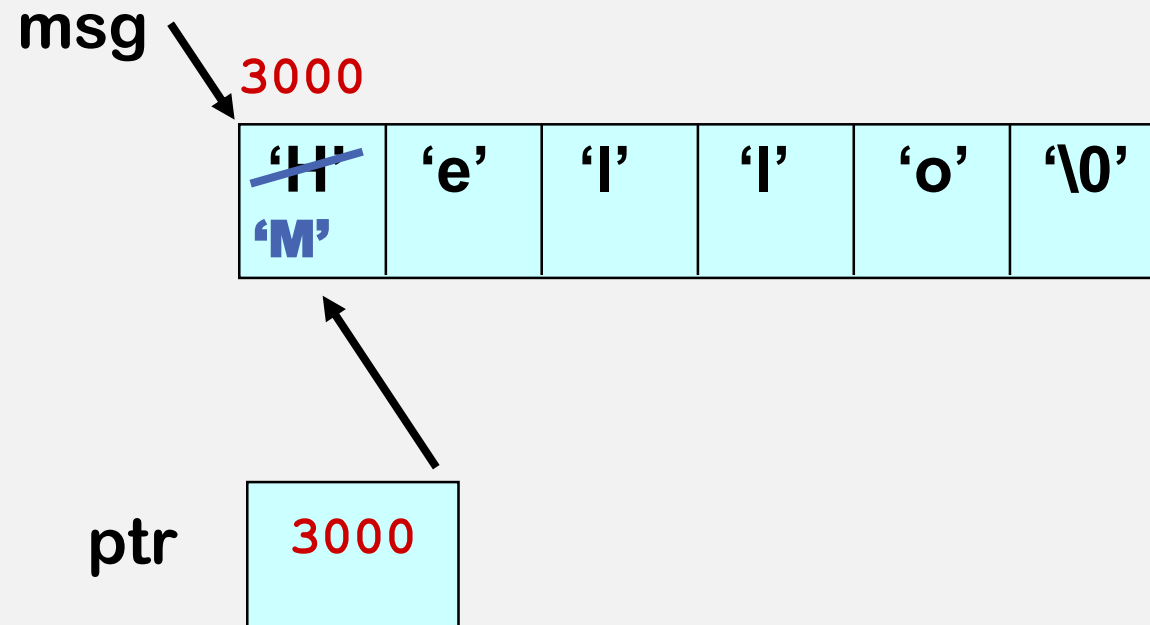
```
ptr++;
```

```
*ptr = 'a';
```



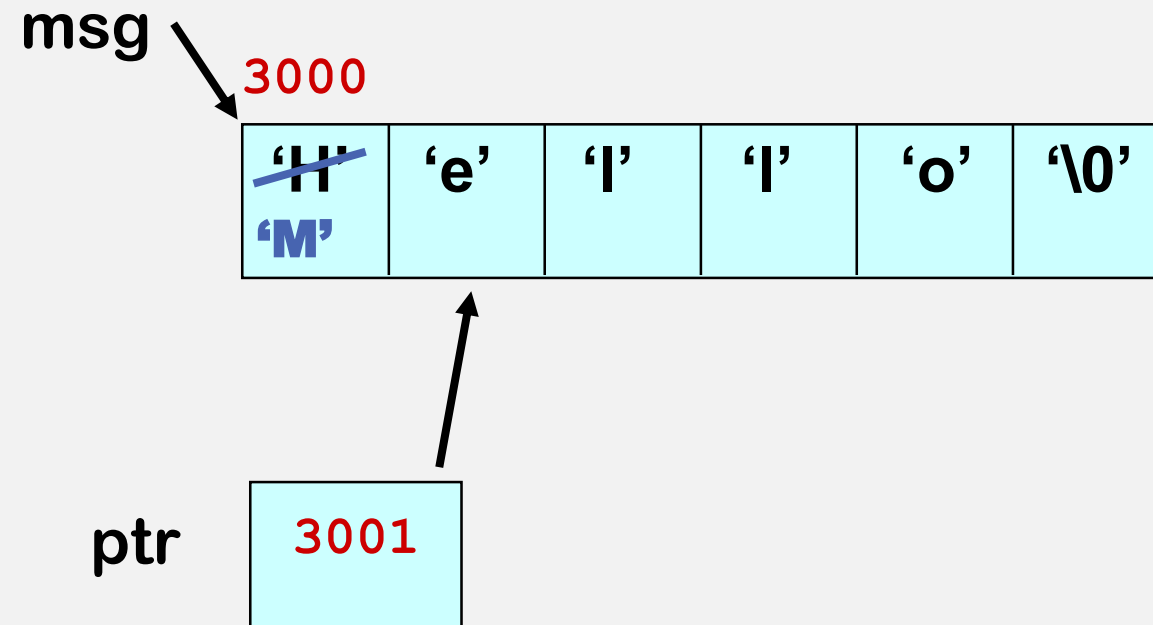
Using a Pointer to Access the Elements of a String

```
char    msg[] = "Hello";  
char*   ptr;  
ptr     = msg;  
→ *ptr  = 'M' ;  
ptr++;  
  
*ptr = 'a';
```



Using a Pointer to Access the Elements of a String

```
char    msg[] = "Hello";  
char*   ptr;  
ptr     = msg;  
*ptr    = 'M' ;  
→ ptr++;  
  
*ptr = 'a';
```



Using a Pointer to Access the Elements of a String

```
char    msg[] = "Hello";
```

```
char*   ptr;
```

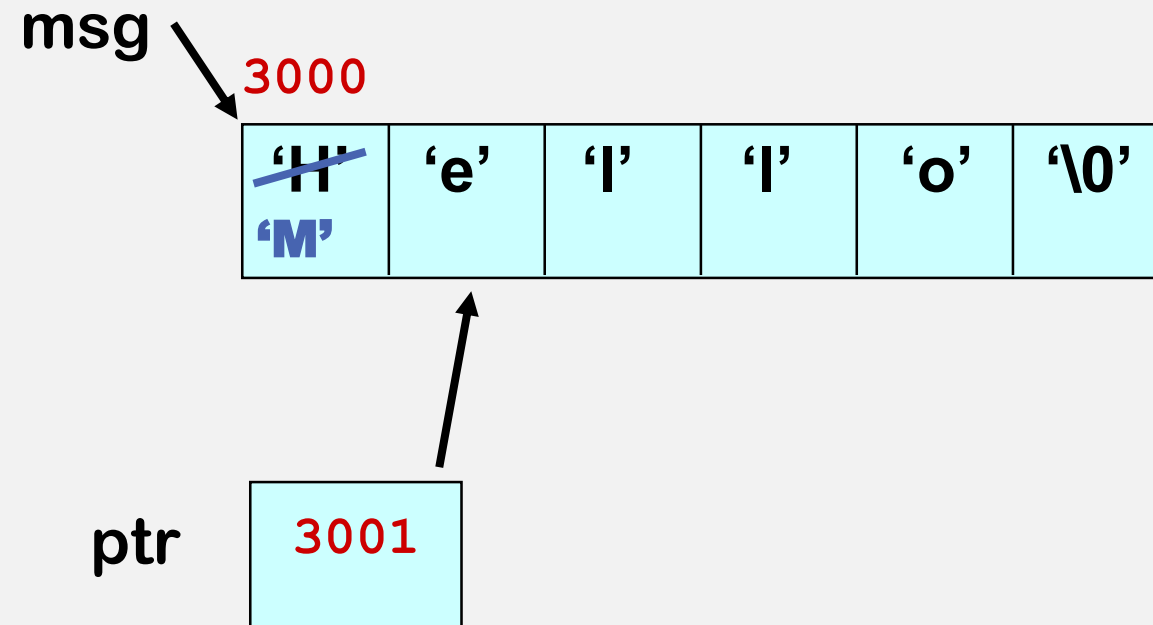
```
ptr     = msg;
```

```
*ptr    = 'M' ;
```

```
ptr++;
```

➔

```
*ptr = 'a';
```



Using a Pointer to Access the Elements of a String

```
char    msg[] = "Hello";
```

```
char*   ptr;
```

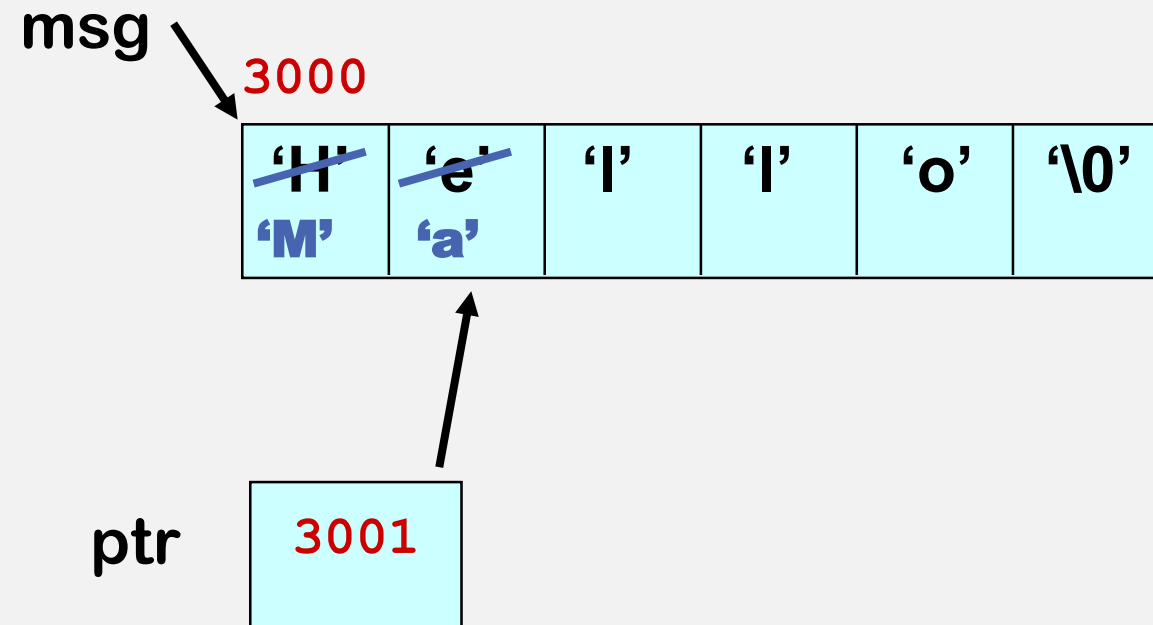
```
ptr     = msg;
```

```
*ptr    = 'M' ;
```

```
ptr++;
```

➔

```
*ptr = 'a';
```



Reference Variables in C++, but not in C

- Reference variable = alias for another variable
 - Contains the address of a variable (like a pointer)
 - No need to perform any dereferencing (unlike a pointer)
 - Must be initialized when it is declared



```
int x = 5;
int &z = x;           // z is another name for x
int &y ;              // Error: reference must be initialized
cout << x << endl;   // prints 5
cout << z << endl;   // prints 5
z = 9;               // same as x = 9;
cout << x << endl;   // prints 9
cout << z << endl;   // prints 9
```

Why Reference Variables

- Are primarily used as function parameters
- Advantages of using references:
 - you don't have to pass the address of a variable
 - you don't have to dereference the variable inside the called function

Why Reference Variables

- Are primarily used as function parameters
- Advantages of using references:
 - you don't have to pass the address of a variable
 - you don't have to dereference the variable inside the called function

```
void swap(int *a, int *b) {  
    int temp = *a;        // (2)  
    *a = *b;              // (3)  
    *b = temp;  
}
```

```
void swap(int& a, int& b) {  
    int temp = a;          // (2)  
    a = b;                 // (3)  
    b = temp;  
}
```

C++ For C Coders 4

Data Structures
C++ for C Coders

한동대학교 김영섭 교수
idebtor@gmail.com

pointer
address-of operator
reference & dereference operators