Power C++

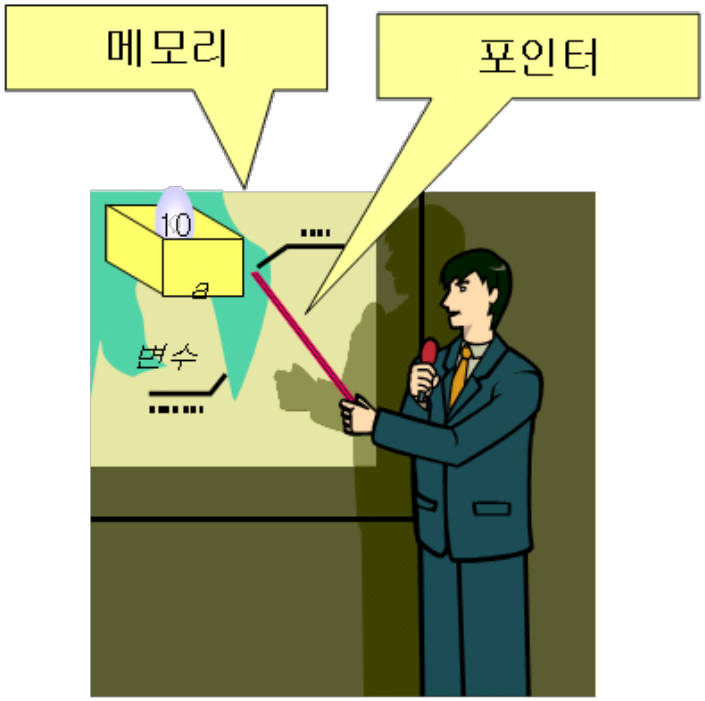
제6장 포인터와 문자열

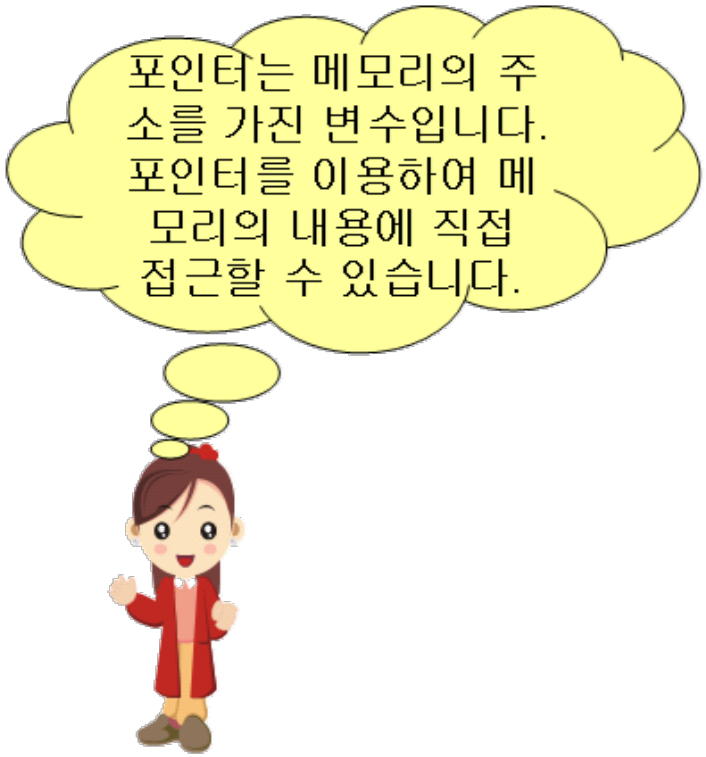
이번 장에서 학습할 내용

- 포인터이란?
- 변수의 주소
- 포인터의 선언
- 간접 참조 연산자
- 포인터 연산
- 포인터와 배열
- 포인터와 함수
- 문자 표현 방법
- 문자열 표현 방법
- 문자열이란 무엇인가?
- 문자열의 입출력
- 문자처리 라이브러리 함수

포인터란?

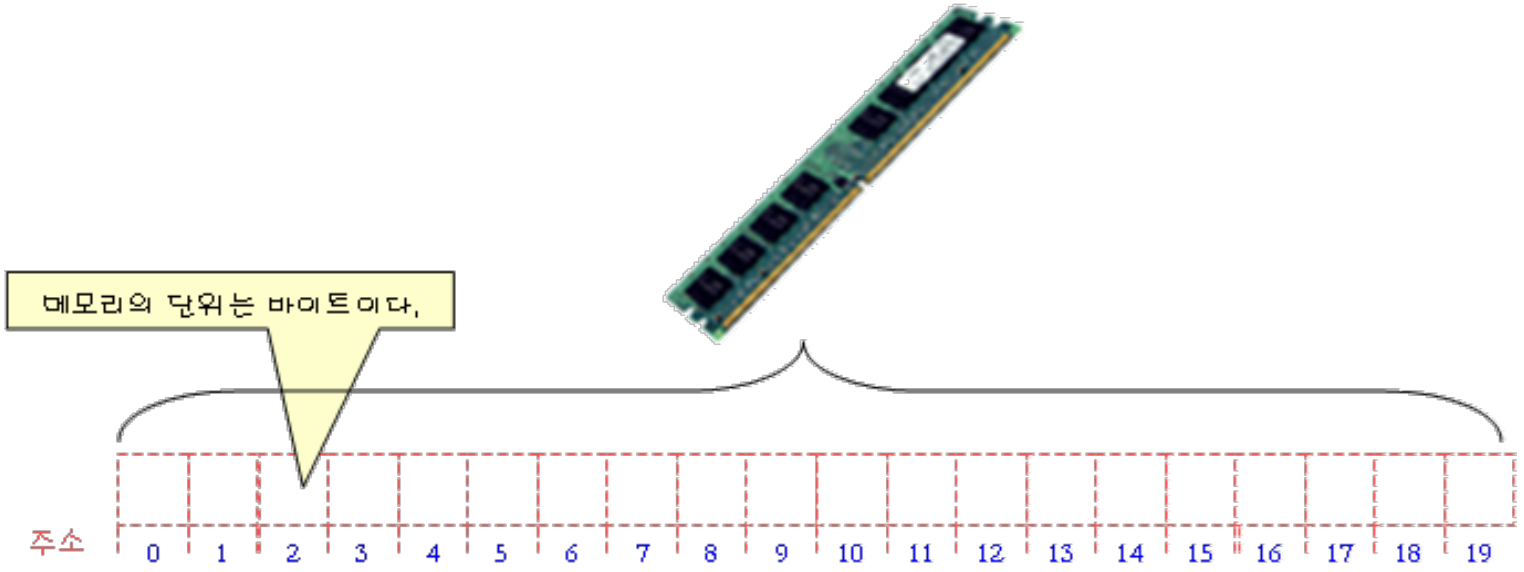
- *포인터(pointer):* 주소를 가지고 있는 변수





메모리의 구조

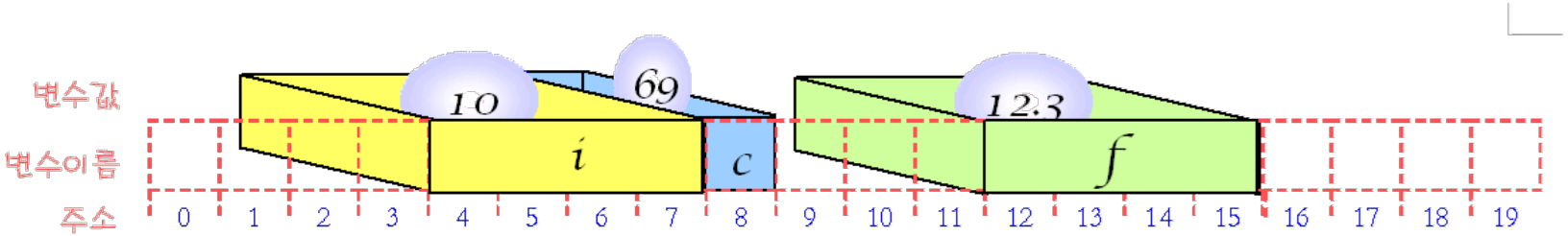
- 변수는 메모리에 저장된다.
- 메모리는 바이트 단위로 액세스된다.
  - 첫번째 바이트의 주소는 0, 두번째 바이트는 1,...



변수와 메모리

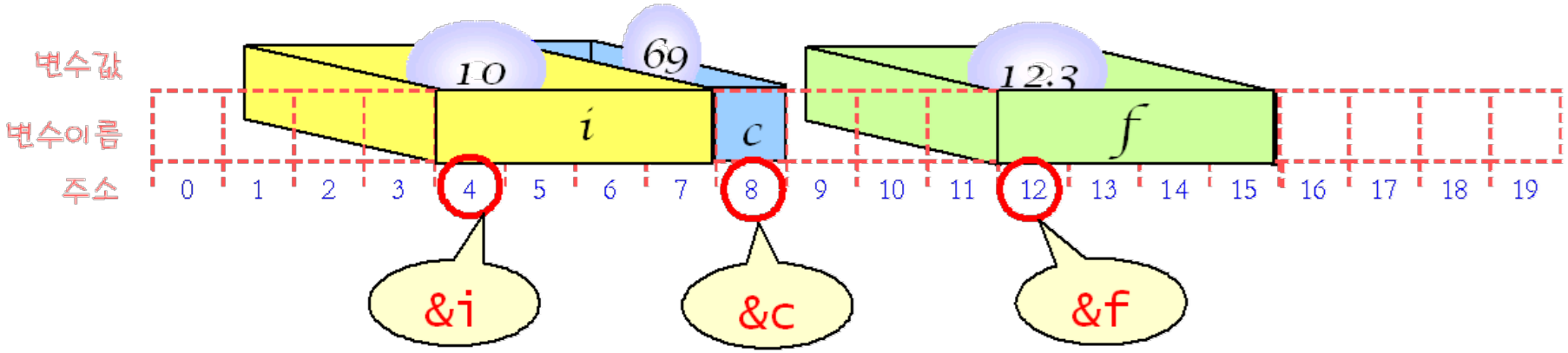
- 변수의 크기에 따라서 차지하는 메모리 공간이 달라진다.
- char형 변수: 1바이트, int형 변수: 4바이트,...

```
int main()
{
  int i = 10;
  char c = 69;
  float f = 12.3;
}
```



변수의 주소

- 변수의 주소를 계산하는 연산자: &
- 변수 i의 주소: &i



변수의 주소

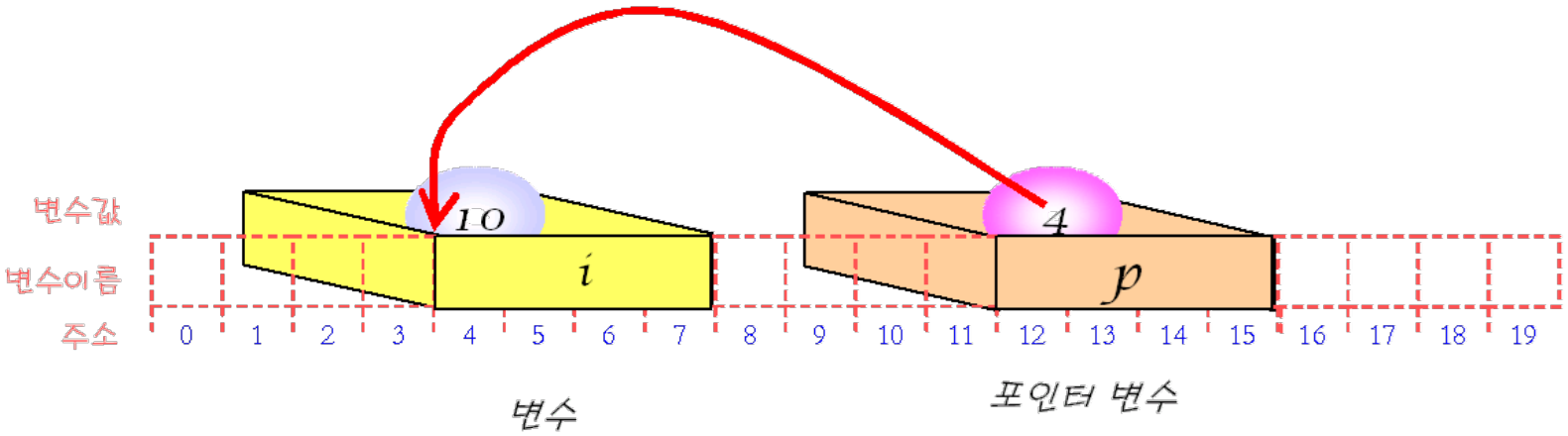
```
#include <iostream>
using namespace std;
int main()
{
	int i = 10;
	char c = 69;
	double f = 12.3;

	cout << "i의 주소: " << &i << endl;		// 변수 i의 주소출력
	cout << "c의 주소: " << (void *)&c << endl;		// 변수 c의 주소출력
	cout << "f의 주소: " << &f << endl;		// 변수 f의 주소출력
	return 0;
}
```

```
i의 주소: 0012FF7C
c의 주소: 0012FF78
f의 주소: 0012FF70
```

포인터의 선언

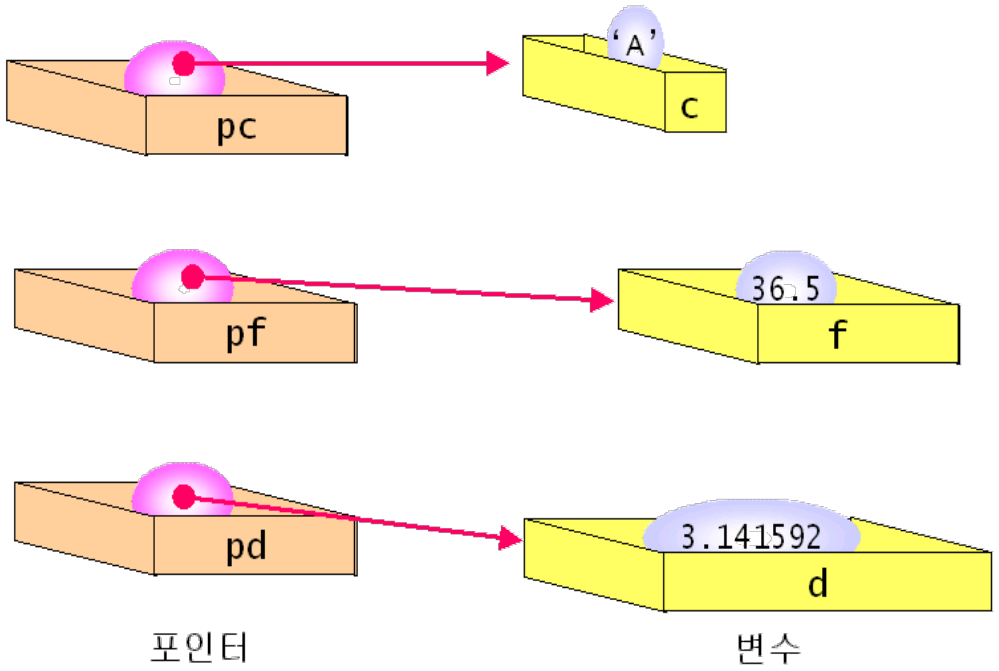
- 포인터: 변수의 주소를 가지고 있는 변수

```
int i = 10;          // 정수형 변수 i 선언
int *p = &i;         // 변수 i의 주소가 포인터 p로 대입
```



다양한 포인터의 선언

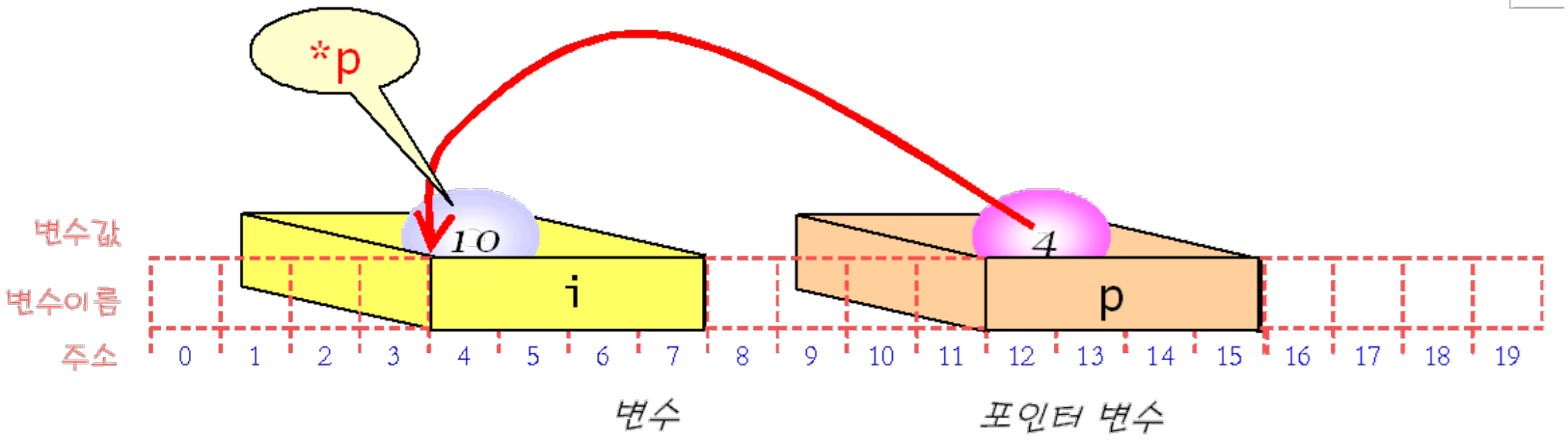
```
char c = 'A';                    // 문자형 변수 c
float f = 36.5;                  // 실수형 변수 f
double d = 3.141592;             // 실수형 변수 d

char *pc = &c;                   // 문자를 가리키는 포인터 pc
float *pf = &f;                  // 실수를 가리키는 포인터 pf
double *pd = &d;                 // 실수를 가리키는 포인터 pd
```



간접 참조 연산자

- 간접 참조 연산자 *: 포인터가 가리키는 값을 가져오는 연산자

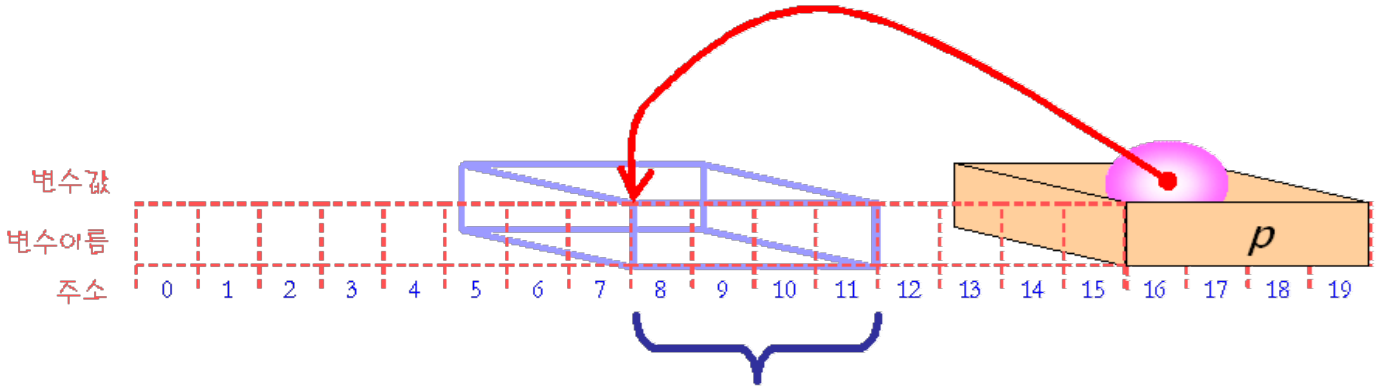
```
int i = 10;
int *p = &i;

cout << *p; // 10이 출력된다.

*p = 20;
cout << *p; // 20이 출력된다.
```

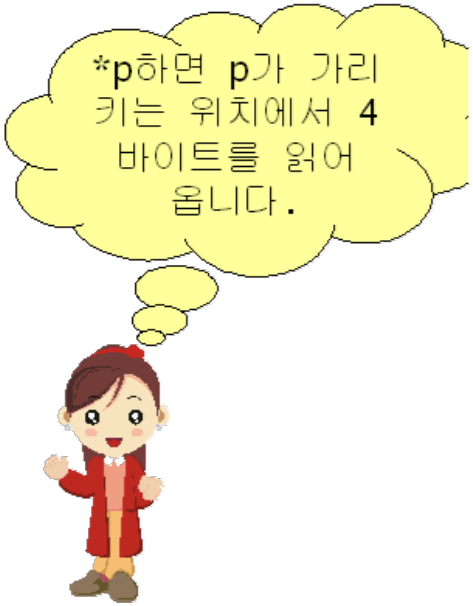


간접 참조 연산자의 해석

- 간접 참조 연산자: 지정된 위치에서 포인터의 타입에 따라 값을 읽어들인다.

```
int *p = 8;                    // 위치 8에서 정수를 읽는다.
char *pc = 8;                  // 위치 8에서 문자를 읽는다.
double *pd = 8;                // 위치 8에서 실수를 읽는다.
```





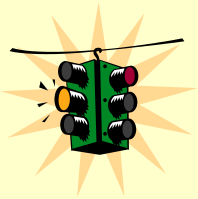
포인터 예제 #1

```
#include <iostream>
using namespace std;
int main()
{
	int i = 3000;
	int *p = &i;                                    // 변수와 포인터 연결

	cout << &i << endl;                 // 변수의 주소 출력
	cout << p << endl; // 포인터의 값 출력

	cout << i << endl;  // 변수의 값 출력
	cout << *p << endl;                 // 포인터를 통한 간접 참조 값 출력
	return 0;
}
```

```
0012FF7C
0012FF7C
3000
3000
```

포인터 예제 #2

```
#include <iostream>
using namespace std;
int main()
{
	char c = 'A';				// 문자형 변수 정의
	int i = 10000;		// 정수형 변수 정의
	double d = 6.78;	// 실수형 변수 정의

	char *pc = &c;		// 문자형 포인터 정의 및 초기화
	int *pi = &i;		// 정수형 포인터 정의 및 초기화
	double *pd = &d;	// 실수형 포인터 정의 및 초기화

	(*pc)++;				// 간접 참조로 1 증가
	*pi = *pi + 1;		// 간접 참조로 1 증가
	*pd += 1;				// 간접 참조로 1 증가

	cout << c << endl;
	cout << i << endl;
	cout << d << endl;
	return 0;
}
```



*pc++라고 하면 안됨

++가 우선순위

```
c = B
i = 10001
d = 7.780000
```

포인터 사용시 주의점 #1

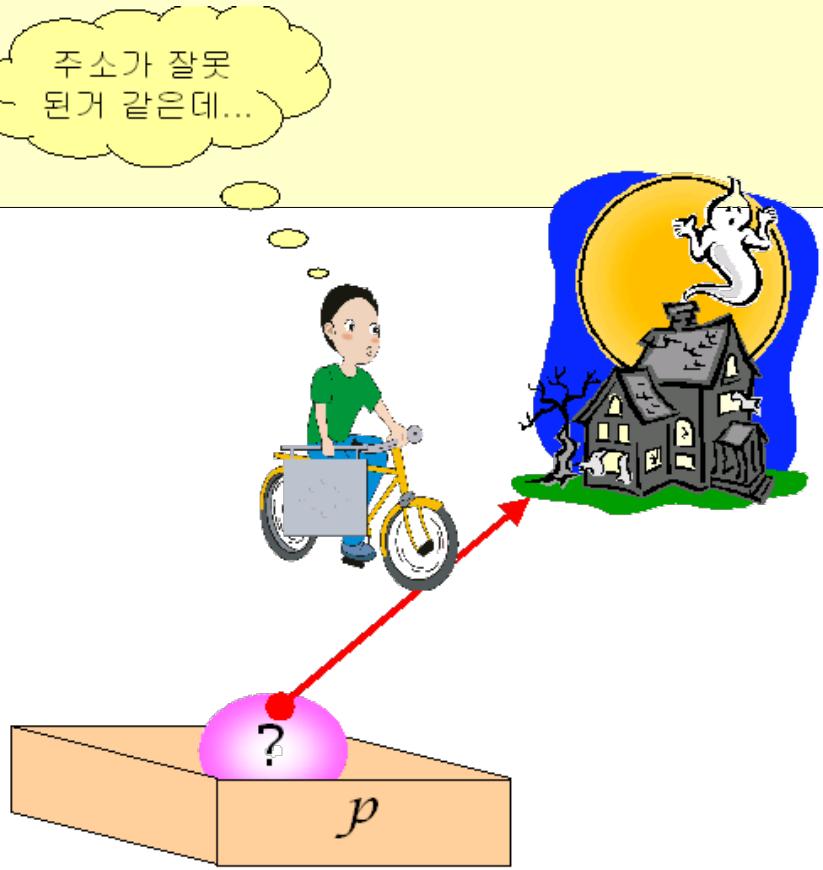
- 포인터의 타입과 변수의 타입은 일치하여야 한다.

```
#include <stdio.h>

int main(void)
{
	int i;
	double *pd;

	pd = &i;		// 오류! double형 포인터에 int형 변수의 주소를 대입
	*pd = 36.5;

	return 0;
}
```

포인터 사용시 주의점 #2

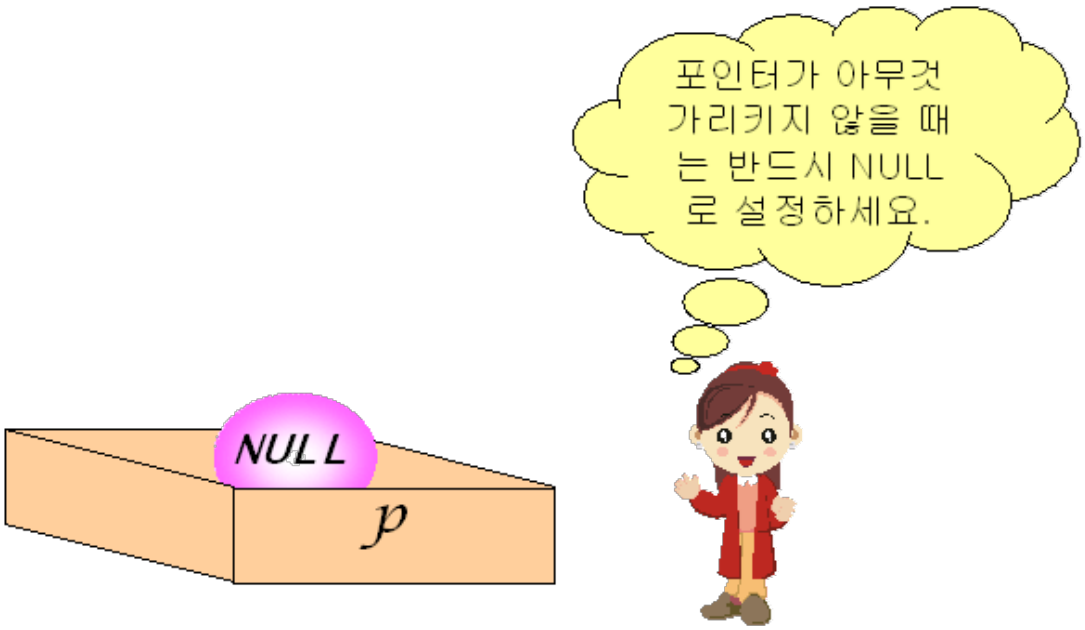
- 초기화가 안된 포인터를 사용하면 안된다.

```
int main(void)
{
    int *p; // 포인터 p는 초기화가 안되어 있음

    *p = 100;          // 위험한 코드
    return 0;
}
```



포인터 사용시 주의점 #3

- 포인터가 아무것도 가리키고 있지 않는 경우에는 NULL로 초기화
- NULL 포인터를 가지고 간접 참조하면 하드웨어로 감지할 수 있다.
- 포인터의 유효성 여부 판단이 쉽다.



포인터 연산

- 가능한 연산: 증가, 감소, 덧셈, 뺄셈 연산
- 증가 연산의 경우 증가되는 값은 포인터가 가리키는 객체의 크기

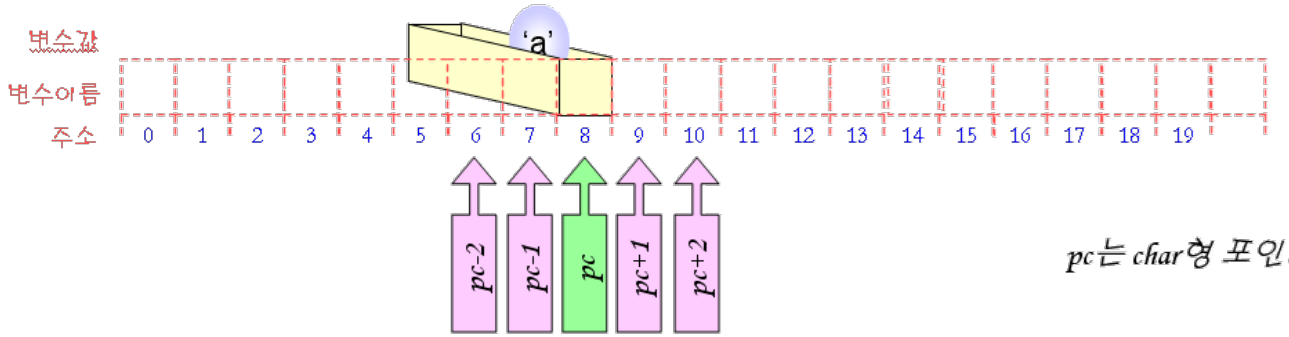
포인터 타입	*++연산후 증가되는값*
char	***1***
short	***2***
int	***4***
float	***4***
double	***8***

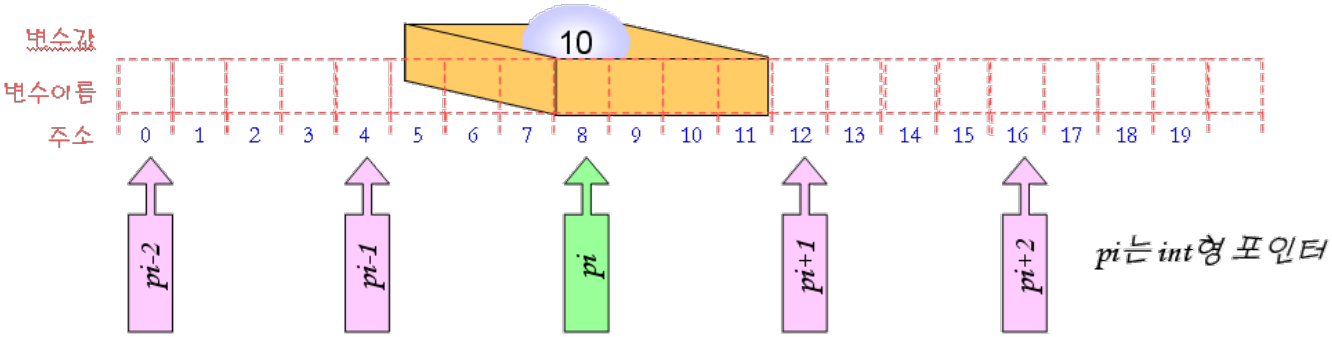
증가 연산 예제

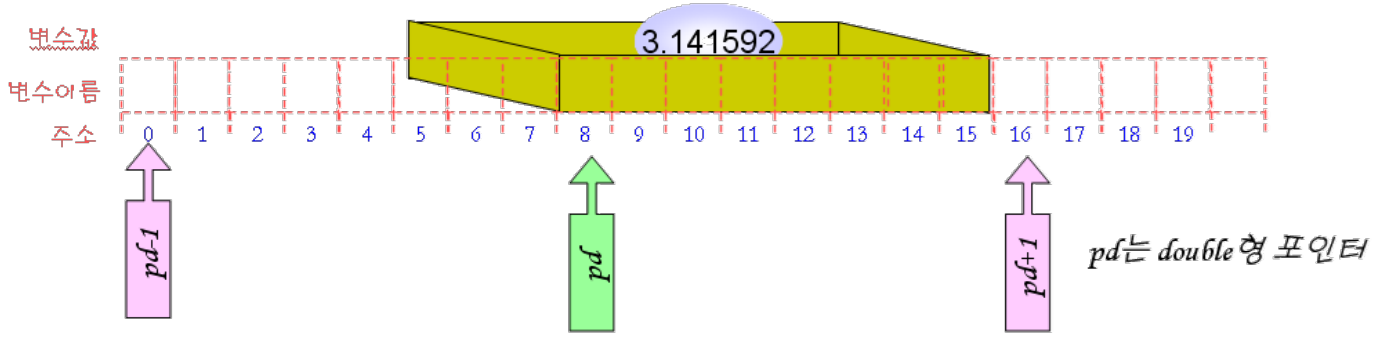
```
#include <iostream>
using namespace std;
int main()
{
	char *pc;
	int *pi;
	double *pd;

	pc = (char *)10000;
	pi = (int *)10000;
	pd = (double *)10000;
	cout << "증가 전 pc = " << (void *)pc << " pi = " << pi << " pd = " << pd << endl;

	pc++;
	pi++;
	pd++;
	cout << "증가 후 pc = " <<  (void *)pc << " pi = " << pi << " pd = " << pd << endl;

	return 0;
}
```

```
증가 전 pc = 00002710 pi = 00002710 pd = 00002710
증가 후 pc = 00002711 pi = 00002714 pd = 00002718
```

포인터의 증감 연산







간접 참조 연산자와 증감 연산자

수식	의미
v = *p++	p가 가리키는 값을 v에 대입한 후에 p를 증가한다.
v = (*p)++	p가 가리키는 값을 v에 대입한 후에 가리키는 값을 증가한다.
v = *++p	p를 증가시킨 후에 p가 가리키는 값을 v에 대입한다.
v = ++*p	p가 가리키는 값을 가져온 후에 그 값을 증가하여 v에 대입한다.

```
// 포인터의 증감 연산
#include <iostream>
using namespace std;

int main()
{
	int i = 10;
	int *pi = &i;

	cout << "i = " << i << " pi = " << pi << endl;
	(*pi)++;
	cout << "i = " << i << " pi = " << pi << endl;
	*pi++;
	cout << "i = " << i << " pi = " << pi << endl;

	return 0;
}
```

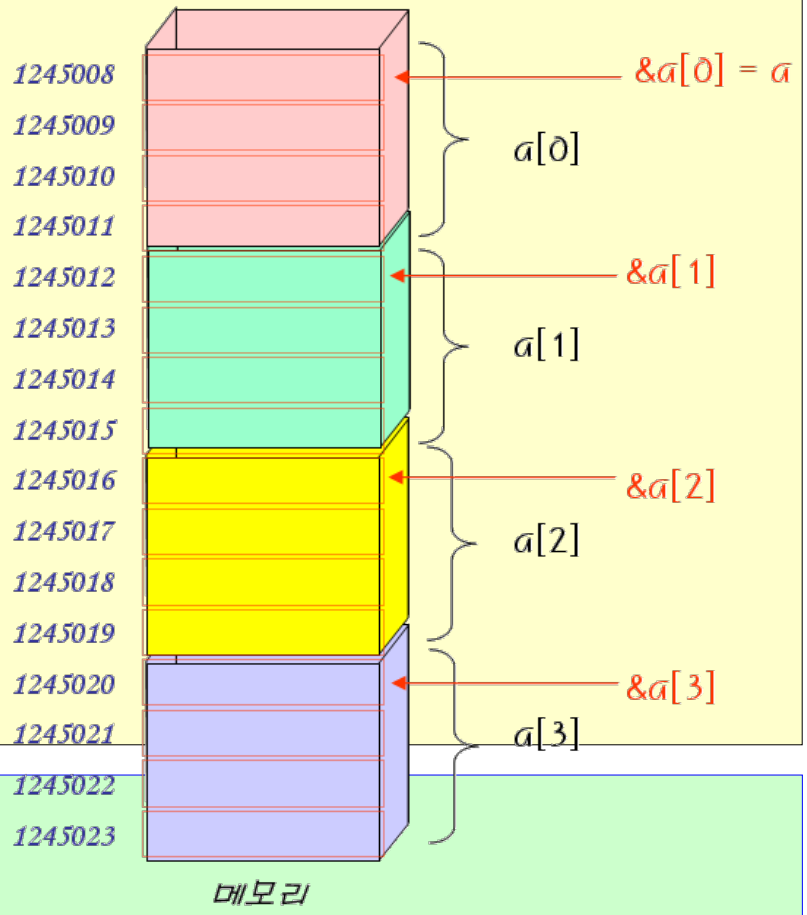
```
i = 10 pi = 0012FF7C
i = 11 pi = 0012FF7C
i = 11 pi = 0012FF80
```

중간 점검 문제

1. 포인터에 대하여 적용할 수 있는 연산에는 어떤 것들이 있는가?
2. int형 포인터 p가 80번지를 가리키고 있었다면 (p+1)은 몇 번지를 가리키는가?
3. double형 배열을 가정하자. 2개의 double형 포인터 p, q가 있고 p가 배열의 2번째 원소를 가리키고 있고 q가 배열의 5번째 원소를 가리키고 있었다면 (q - p)는 얼마인가?
4. p가 포인터라고 하면 *p++와 (*p)++의 차이점은 무엇인가?
5. p가 포인터라고 하면 *(p+3)의 의미는 무엇인가?

포인터와 배열

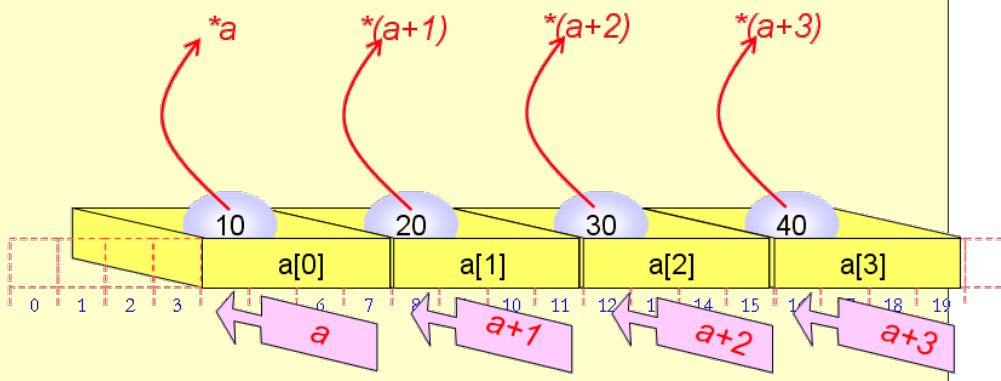
```
#include <iostream>
using namespace std;
int main()
{
	int a[] = { 10, 20, 30, 40, 50 };

	cout << "&a[0] = " << &a[0] << endl;
	cout << "&a[1] = " << &a[1] << endl;
	cout << "&a[2] = " << &a[2] << endl;

	cout << "a = " << a << endl;

	return 0;
}
```




```
&a[0] = 1245008
&a[1] = 1245012
&a[2] = 1245016
a = 1245008
```

포인터와 배열

```
#include <iostream>
using namespace std;
int main()
{
      int a[] = { 10, 20, 30, 40, 50 };

      cout << "a = " << a << endl;
      cout << "a + 1 = " << a+1 << endl;
      cout << "*a = " << *a << endl;
      cout << "*(a+1) " << *(a+1) << endl;
      return 0;
}
```




```
a = 1245008
a + 1 = 1245012
*a = 10
*(a+1) = 20
```

포인터를 사용한 방법의 장점

- 인덱스 표기법보다 빠르다.
  - 원소의 주소를 계산할 필요가 없다.

```
int get_sum1(int a[], int n)
{
      int i;
      int sum = 0;

      for(i = 0; i < n; i++ )
            sum += a[i];
      return sum;
}
```

인덱스 표기법 사용

```
int get_sum2(int a[], int n)
{
      int i;
      int *p;
      int sum = 0;

      p = a;
      for(i = 0; i < n; i++ )
            sum += *p++;
      return sum;
}
```

포인터 사용

중간 점검 문제

1. 배열의 첫 번째 원소의 주소를 계산하는 2가지 방법을 설명하라.
2. 배열 a[]에서 *a의 의미는 무엇인가?
3. 배열의 이름에 다른 변수의 주소를 대입할 수 있는가?
4. 포인터를 이용하여 배열의 원소들을 참조할 수 있는가?
5. 포인터를 배열의 이름처럼 사용할 수 있는가?

동적 할당 메모리의 개념

- 프로그램이 메모리를 할당받는 방법
  - 정적(static)
  - 동적(dynamic)

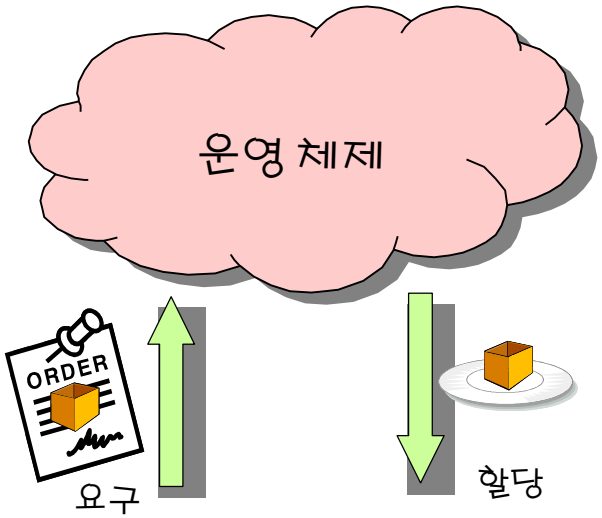
- 정적 메모리 할당
  - 프로그램이 시작되기 전에 미리 정해진 크기의 메모리를 할당받는 것
  - 메모리의 크기는 프로그램이 시작하기 전에 결정

```
int i, j;
int buffer[80];
char name[] = “data structure";
```

  - 처음에 결정된 크기보다 더 큰 입력이 들어온다면 처리하지 못함
  - 더 작은 입력이 들어온다면 남은 메모리 공간은 낭비

동적 메모리

- 동적 메모리
  - 실행 도중에 동적으로 메모리를 할당받는 것
  - 사용이 끝나면 시스템에 메모리를 반납
  - 필요한 만큼만 할당을 받고 메모리를 매우 효율적으로 사용
  - new와 delete 키워드 사용

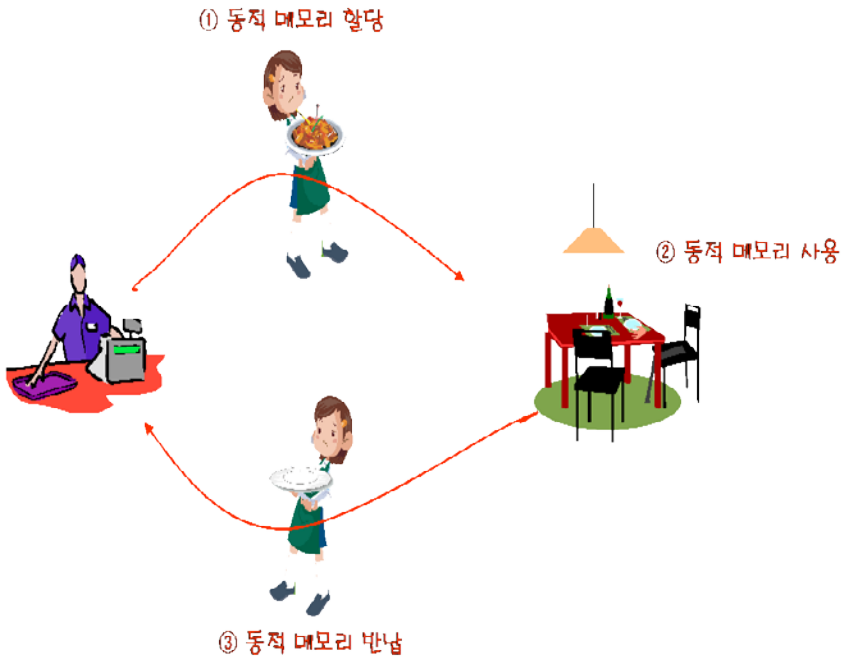



```
#include <iostream>
using namespace std;

int main()
{
 int *p;
 p = new int;
 ...
}
```

프로그램

동적 메모리 할당의 과정

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

int main()
{
        int *pi;        // 동적 메모리를 가리키는 포인터

        pi = new int; // ① 동적 메모리 할당

        *pi = 100;              // ② 동적 메모리 사용
        delete pi;              // ③ 동적 메모리 반납

        return 0;
}
```



파생 자료형인 경우

- 배열

```
double *pd = new double[10];
...
delete[] pd;
```

메모리 누수의 예제

```
void sub()
{
    int *pi = new int;          // ①
    *pi = 67;

    pi = new int;               // ②
    *pi = 99;
}
```

잘못된 버전

```
void sub()
{
    int *pi = new int;          // ①
    *pi = 67;
    delete pi;

    pi = new int;      // ②
    *pi = 99;
    delete pi;
}
```

올바른 버전

const 포인터

- const int *p1;
- p1은 const int에 대한 포인터이다. 즉 p1이 가리키는 내용이 상수가 된다.
- *p1 = 100;(X)

- int * const p2;
- 이번에는 정수를 가리키는 p2가 상수라는 의미이다. 즉 p2의 내용이 변경될 수 없다.
- p2 = p1; (X)

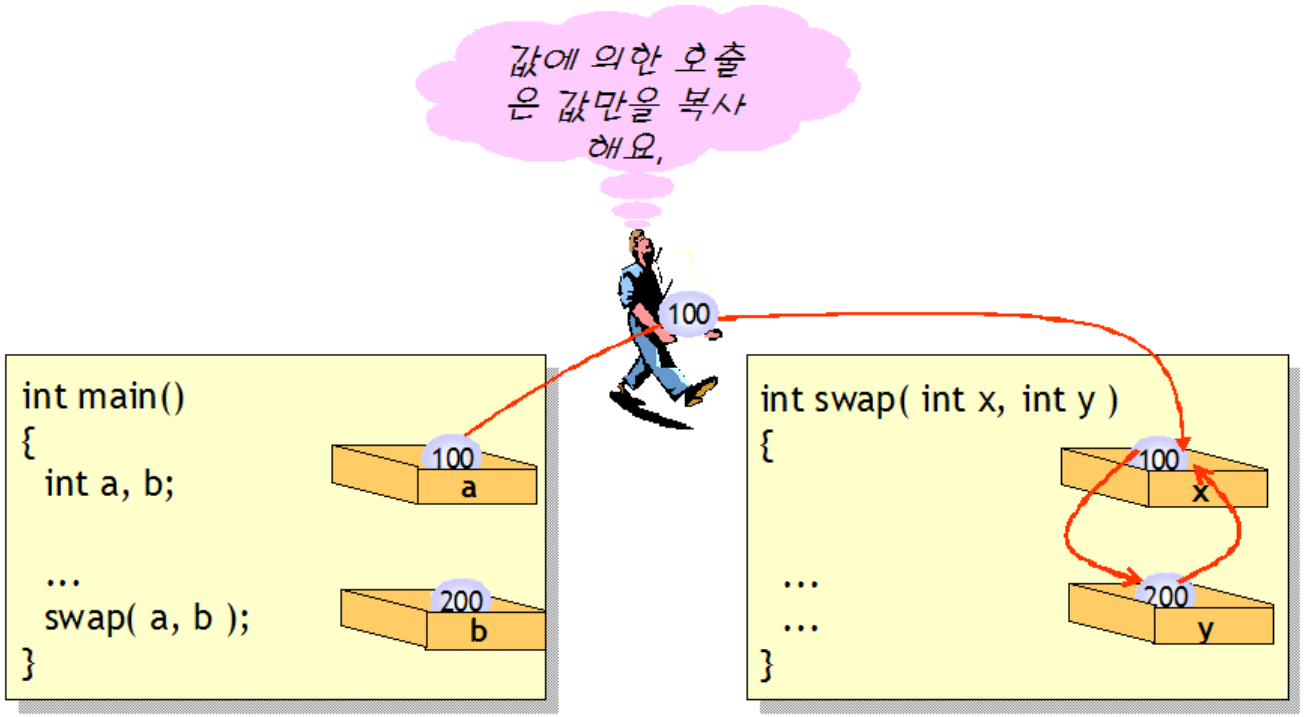
중간 점검 문제

1. 프로그램의 실행 도중에 메모리를 할당받아서 사용하는 것을 _____이라고 한다.
2. 동적으로 메모리를 할당받을 때 사용하는 키워드는 _____이다.
3. 동적으로 할당된 메모리를 해제하는 키워드는 _____이다.

포인터와 함수

- C++에서의 인수 전달 방법
  - 값에 의한 호출 (call-by-value)
    - 함수로 복사본이 전달된다.
  - 참조에 의한 호출 (call-by-reference)
    - 함수로 원본이 전달된다.

값에 의한 호출



swap() 함수 #1

- 변수 2개의 값을 바꾸는 작업을 함수로 작성

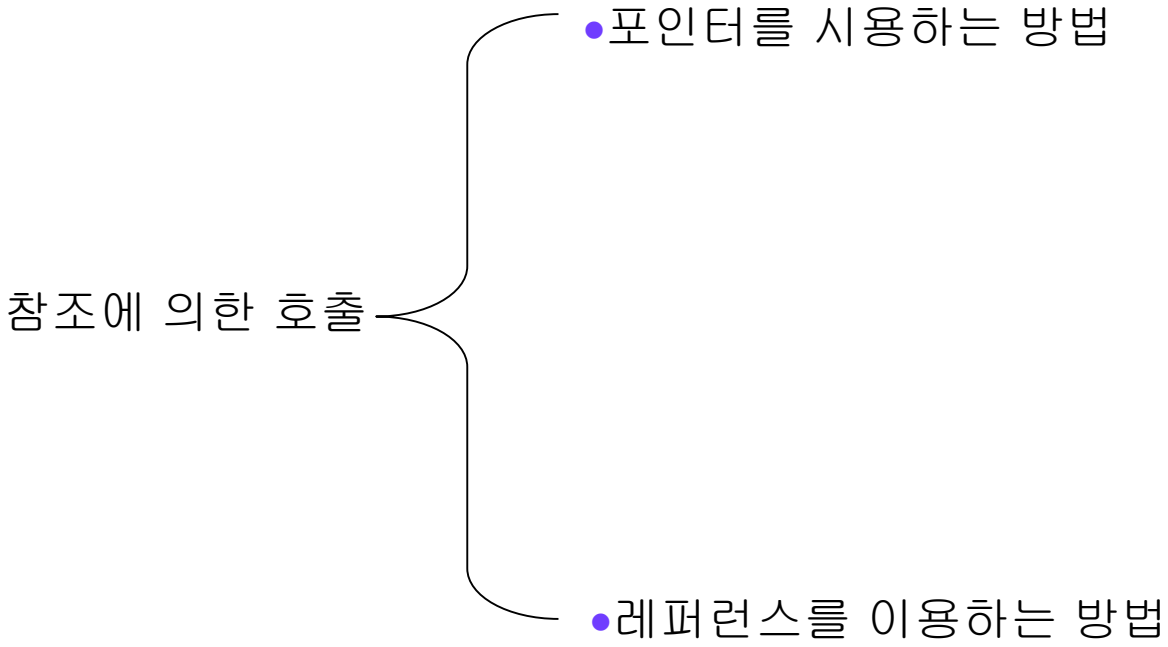
```
#include <iostream>
using namespace std;
void swap(int x, int y);

int main()
{
	int a = 100, b = 200;
	cout << "swap() 호출전: a = " << a << ", b = " << b << endl;
	swap(a, b);
	cout << "swap() 호출후: a = " << a << ", b = " << b << endl;
	return 0;
}
```

swap() 함수 #1

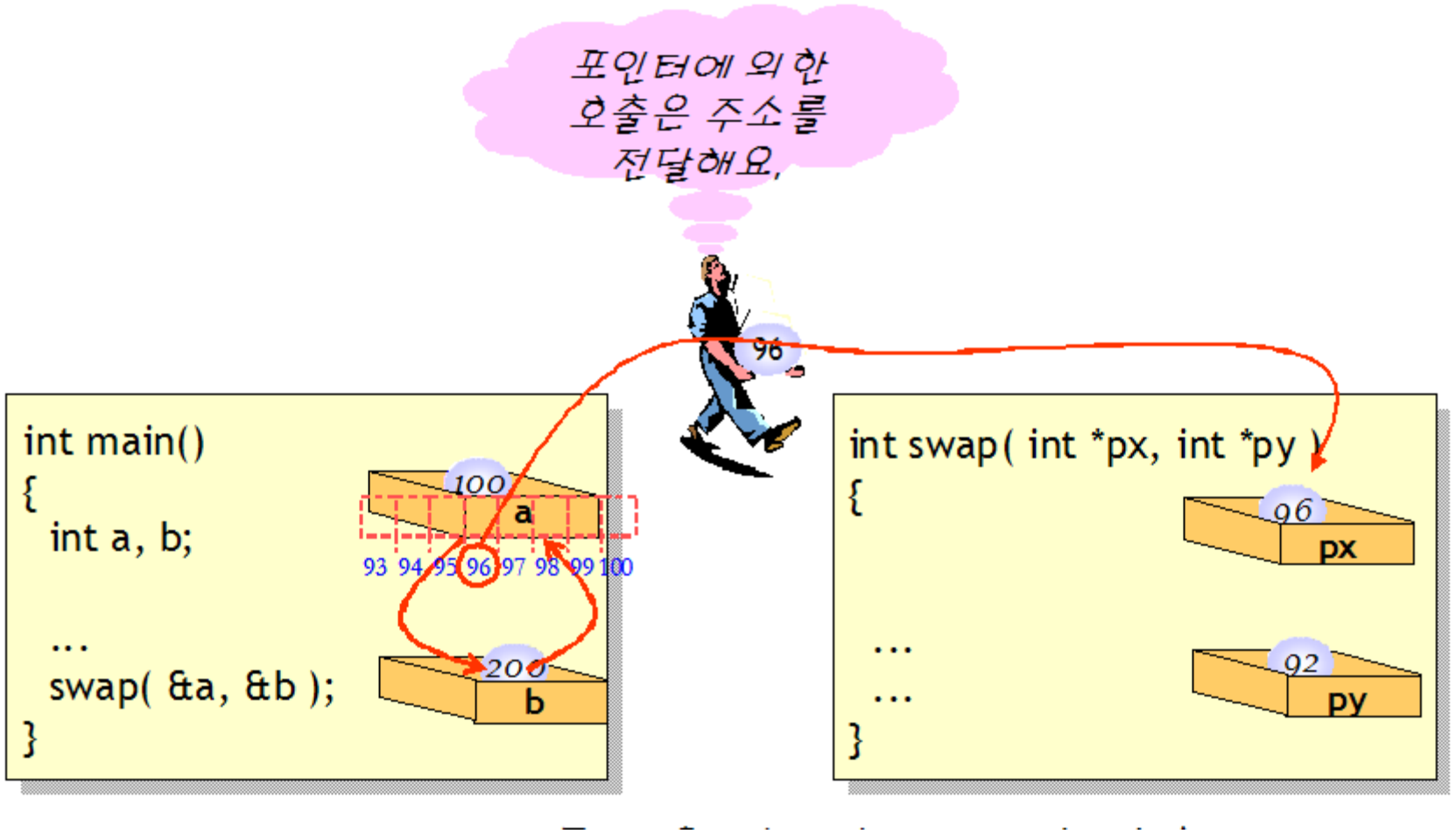
```
void swap(int x, int y)
{
	int tmp;

	cout << "In swap() : x = " << x << ", y = " << y << endl;

	tmp = x; x = y; y = tmp;

	cout << "In swap() : x = " << x << ", y = " << y << endl;
}
```

```
swap() 호출전: a = 100, b = 200
In swap() : x = 100, y = 200
In swap() : x = 200, y = 100
swap() 호출후: a = 100, b = 200
계속하려면 아무 키나 누르십시오 . . .
```

참조에 의한 호출



참조에 의한 호출(포인터 이용)



swap() 함수 #2

- 포인터를 이용

```
void swap(int *px, int *py);

int main()
{
	int a = 100, b = 200;
	cout << "swap() 호출전: a = " << a << ", b = " << b << endl;

	swap(&a, &b);

	cout << "swap() 호출후: a = " << a << ", b = " << b << endl;
	return 0;
}
```

swap() 함수 #2

```
void swap(int *px, int *py)
{
        int tmp;

        cout << "In swap() : *px = " << *px << ", *py = " << *py << endl;

        tmp = *px; *px = *py; *py = tmp;

        cout << "In swap() : *px = " << *px << ", *py = " << *py << endl;
}
```

```
swap() 호출전: a = 100, b = 200
In swap() : *px = 100, *py = 200
In swap() : *px = 200, *py = 100
swap() 호출후: a = 200, b = 100
```

2개 이상의 결과를 반환

```
#include <iostream>
using namespace std;
int get_line_parameter(int x1, int y1, int x2, int y2, float *slope, float *yintercept)
{
        if( x1 == x2 )
                return -1;
        else
        {
          *slope = (float)(y2 - y1)/(float)(x2 - x1);
          *yintercept = y1 - (*slope)*x1;
          return 0;
        }
}
int main()
{
        float s, y;
        if( get_line_parameter(3, 3, 6, 6, &s, &y) == -1 )
                cout << "에러" << endl;
        else
                cout << "기울기는 "<< s << endl << "y절편은 " << y << endl;
        return 0;
}
```

기울기와 y-절편을 인수로 전달

```
기울기는 1
y절편은 0
```

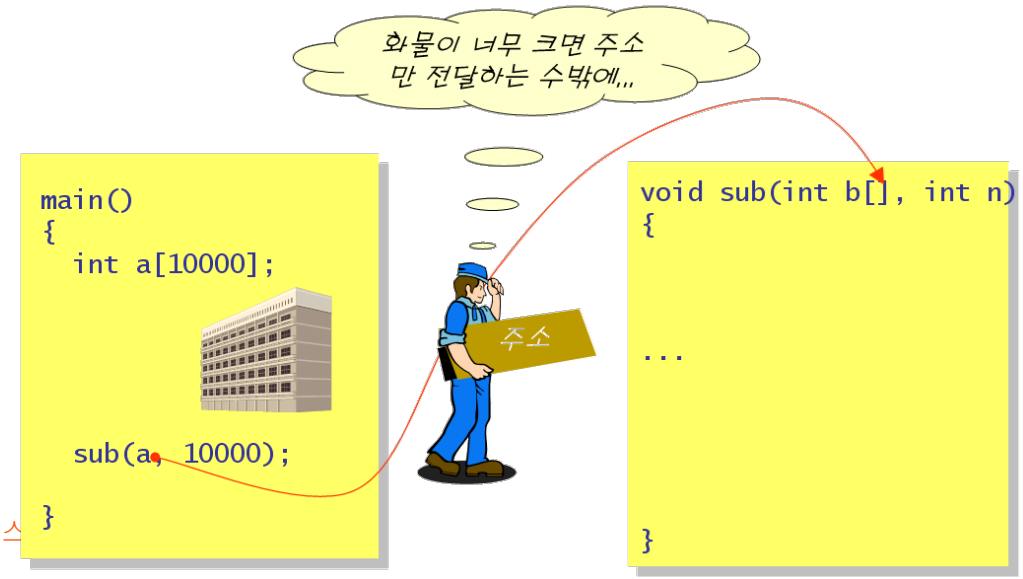
배열이 함수 인수인 경우

- 일반 변수 vs 배열

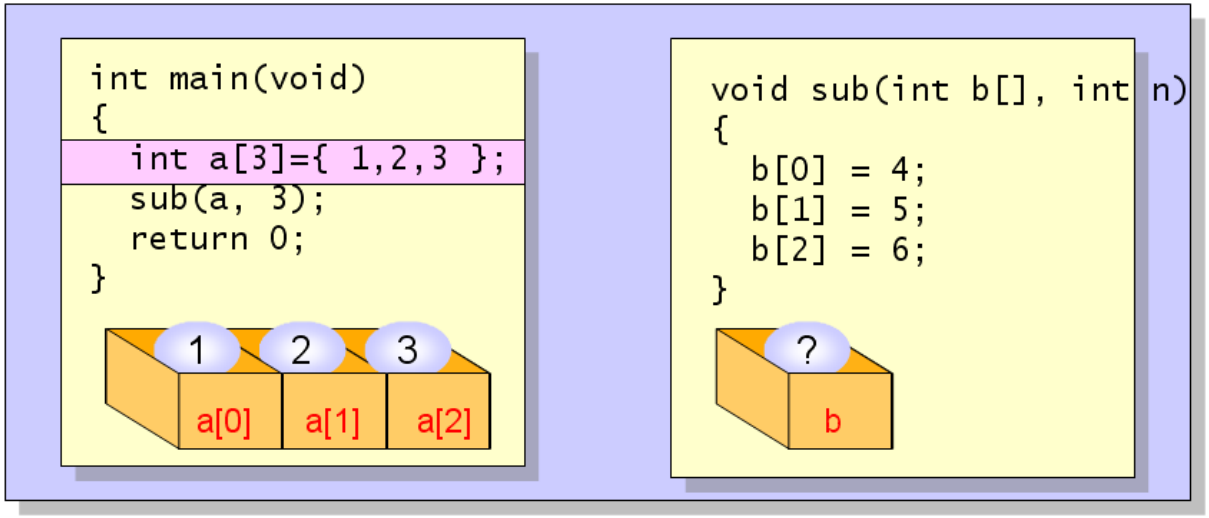
```
// 매개 변수 x에 기억 장소가 할당된다.
void sub(int x)
{
    …
}
```

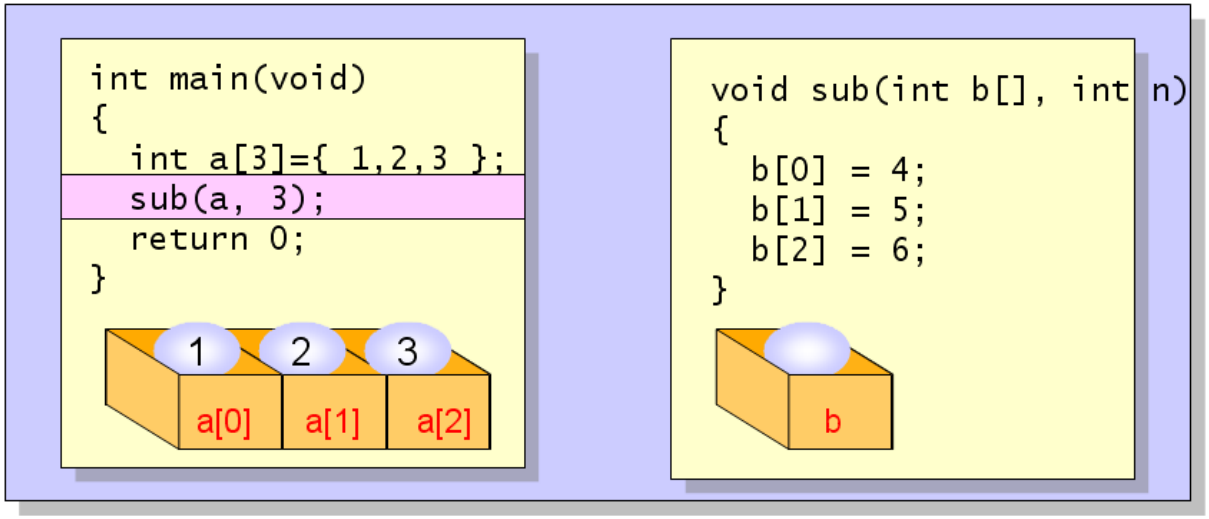
```
// 매개 변수 b[]에 기억 장소가 할당되지 않는다.
void sub(int b[], int n)
{
    …
}
```

- 배열의 경우, 크기가 큰 경우에 복사하려면 많은 시간 소모
- 배열의 경우, 배열의 주소를 전달

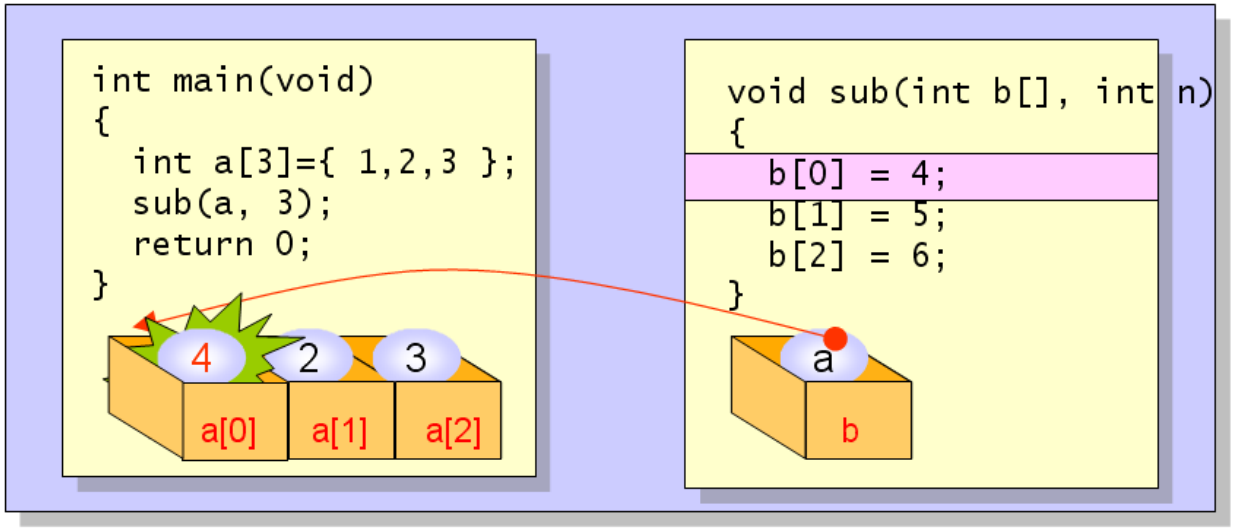


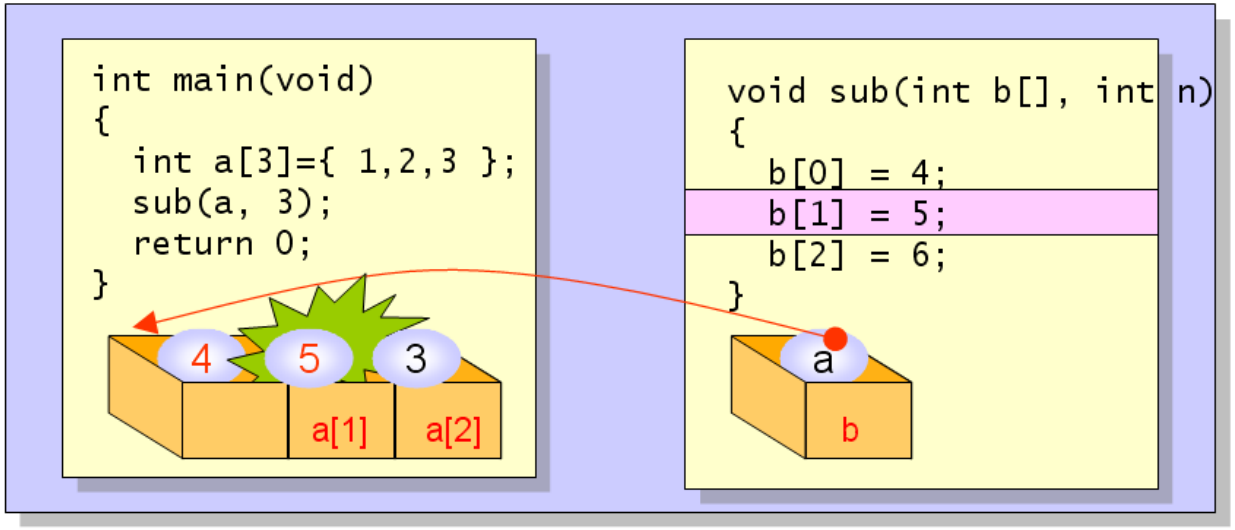
배열이 함수의 인수인 경우 1/3



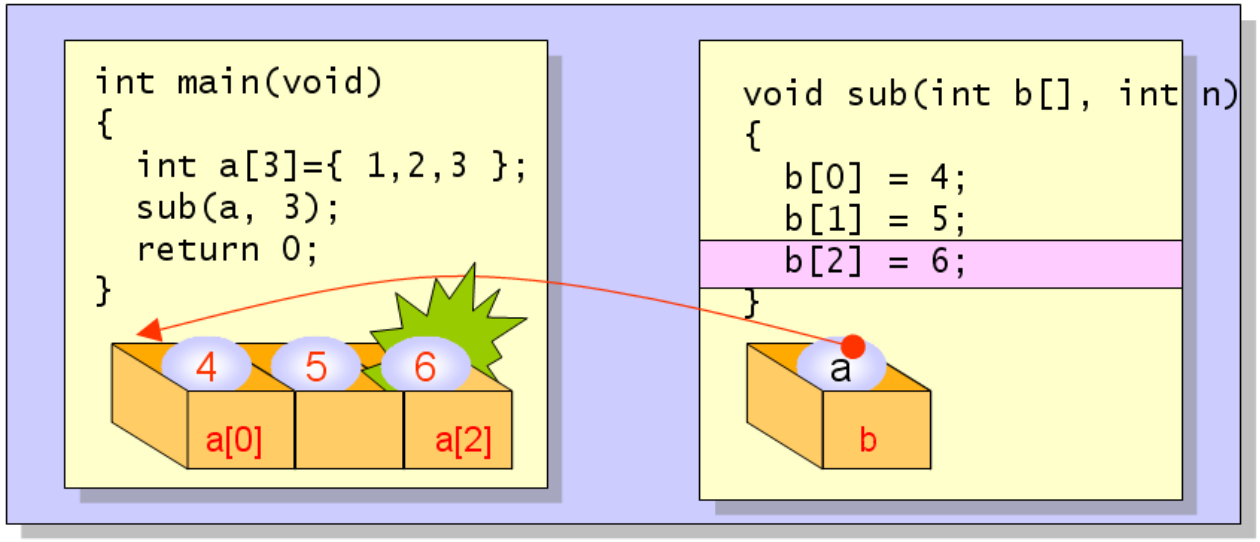


배열이 함수의 인수인 경우 2/3





배열이 함수의 인수인 경우 3/3



주의

- 함수가 종료되더라도 남아 있는 변수의 주소를 반환하여야 한다.
- 지역 변수의 주소를 반환하면 , 함수가 종료되면 사라지기 때문에 오류

```
int *add(int x, int y)
{
      int result;

      result = x + y;
      return &result;
}
```

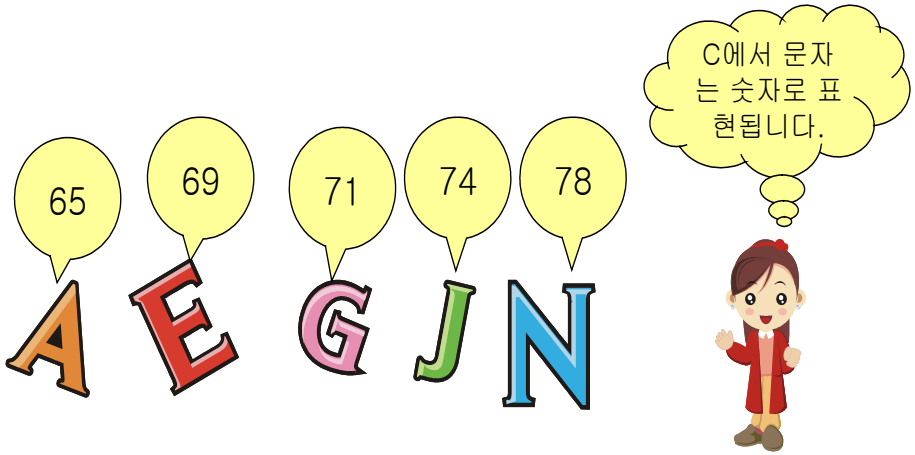
지역 변수 result는 함수가 종료되면 소멸되므로 그 주소를 반환하면 안된다.!!

중간 점검 문제

1. 함수에 매개 변수로 변수의 복사본이 전달되는 것을 _____라고 한다.
2. 함수에 매개 변수로 변수의 원본이 전달되는 것을 _____라고 한다.
3. 배열을 함수의 매개 변수로 지정하는 경우, 배열의 복사가 일어나는가?

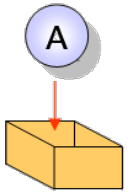
문자표현방법

- 컴퓨터에서는 각각의 문자에 숫자코드를 붙여서 표시한다.
- 아스키코드(ASCII code): 표준적인 8비트 문자코드
  - 0에서 127까지의 숫자를 이용하여 문자표현
- 유니코드(unicode): 표준적인 16비트 문자코드
  - 전세계의 모든 문자를 일관되게 표현하고 다룰 수 있도록 설계

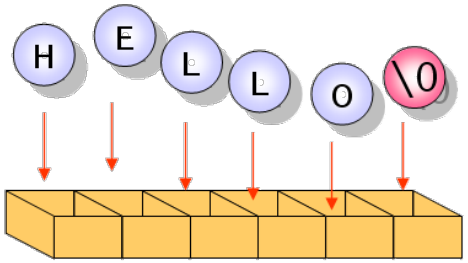


문자열 표현 방법

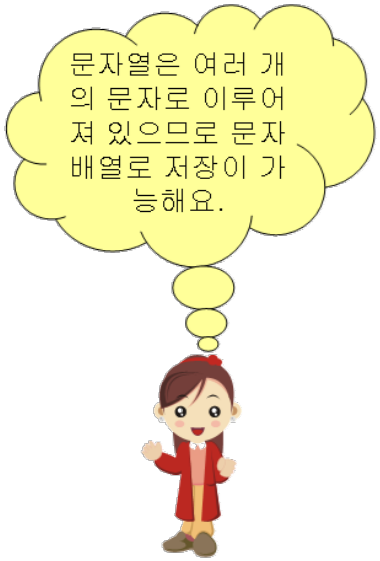
- *문자열(string):* 문자들이 여러 개 모인것
  - "A"
  - "Hello World!"
  - "변수 score의 값은 %d입니다“
- *문자열 상수*
  - "Hello World"
  - "Hong"
  - "string!#$"
  - "guest123"
  - "ascii code = %d“
- *문자열 변수*
  - char형 배열



하나의 문자는 char형 변수로 저장



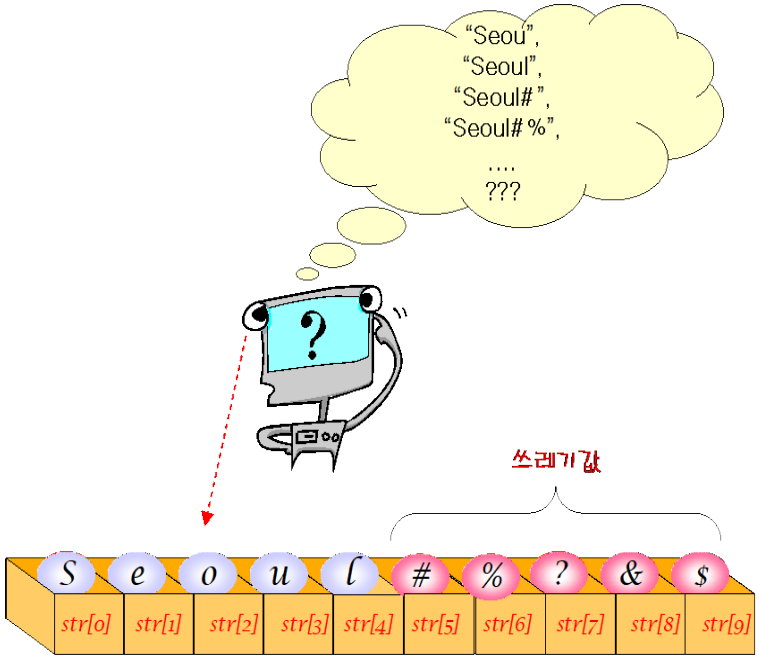
문자열은 char형 배열로 저장



NULL 문자

- NULL 문자: 문자열의 끝을 나타낸다.

- 문자열은 어디서 종료되는지 알수가 없으므로 표시를 해주어야 한다.



문자 배열의 초기화

1. 문자 배열 원소들을 중괄호 안에 넣어주는 방법

   - char str[6] = { 'H', 'e', 'l', 'l', 'o', '\0' };

2. 문자열 상수를 사용하여 초기화하는 방법

   - char str[6] = "Hello";

3. 만약 배열을 크기를 지정하지 않으면 컴파일러가 자동으로 배열의 크기를 초기화값에 맞추어 설정

   - char str[] = "C Bible"; // 배열의 크기는 7이 된다.

문자 배열에 문자를 저장

1. 각각의 문자 배열 원소에 원하는 문자를 개별적으로 대입하는 방법이다.
   - str[0] = 'W';
   - str[1] = 'o';
   - str[2] = 'r';
   - str[3] = 'l';
   - str[4] = 'd';
   - str[5] = '\0';

2. strcpy()를 사용하여 문자열을 문자 배열에 복사
   - strcpy(str, "World");

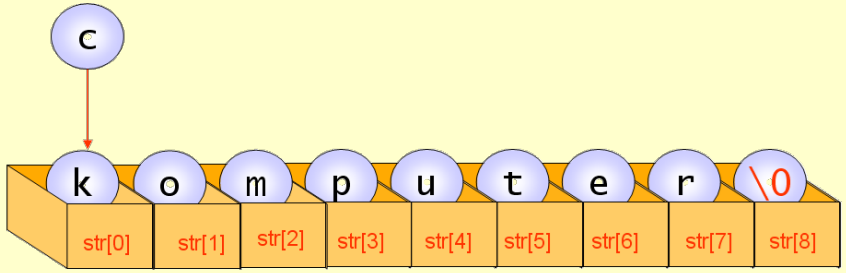
예제 #1

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

int main()
{
	char str1[7] = "Seoul ";
	char str2[3] = { 'i', 's' };
	char str3[] = " the capital city of Korea.";

	cout << str1 << str2 << str3 << endl;
	return 0;
}
```

```
Seoul is the capital city of Korea.
```

예제 #2

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

int main()
{
	char str[] = "komputer";
	int i;

	for(i=0;i<8;i++)
		cout << str[i] << " ";

	str[0] = 'c';
	cout << endl;

	for(i=0;i<8;i++)
		cout << str[i] << " ";
	cout << endl;
	return 0;
}
```




```
k o m p u t e r
c o m p u t e r
```

문자열 길이 계산 예제

```
// 문자열의 길이를 구하는 프로그램
#include <iostream>
using namespace std;

int main()
{
	char str[30] = "C++ language is easy";
	int i = 0;

	while(str[i] != 0)
		i++;
	cout << "문자열 "<< str << "의 길이는 " << i << "입니다." << endl;

	return 0;
}
```

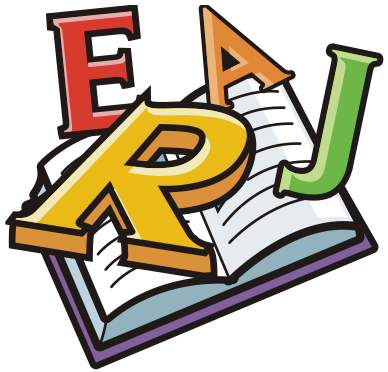
```
문자열 C++ language is easy의 길이는 20입니다.
계속하려면 아무 키나 누르십시오 . . .
```

문자 처리 라이브러리 함수

- 문자를 검사하거나 문자를 변환한다.

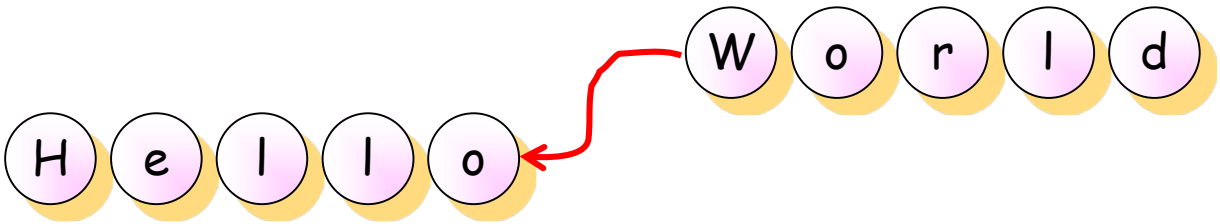
함수	설명
isalpha(c)	c가 영문자인가?(a-z, A-Z)
isupper(c)	c가 대문자인가?(A-Z)
islower(c)	c가 소문자인가?(a-z)
isdigit(c)	c가 숫자인가?(0-9)
isalnum(c)	c가 영문자이나 숫자인가?(a-z, A-Z, 0-9)
isxdigit(c)	c가 16진수의 숫자인가?(0-9, A-F, a-f)
isspace(c)	c가 공백문자인가?(‘ ’, ‘\n', '\t', '\v', '\r')
ispunct(c)	c가 구두점 문자인가?
isprint(c)	C가 출력가능한 문자인가?
iscntrl(c)	c가 제어 문자인가?
isascii(c)	c가 아스키 코드인가?

함수	설명
toupper(c)	c를 대문자로 바꾼다.
tolower(c)	c를 소문자로 바꾼다.
toascii(c)	c를 아스키 코드로 바꾼다.



문자열 처리 라이브러리

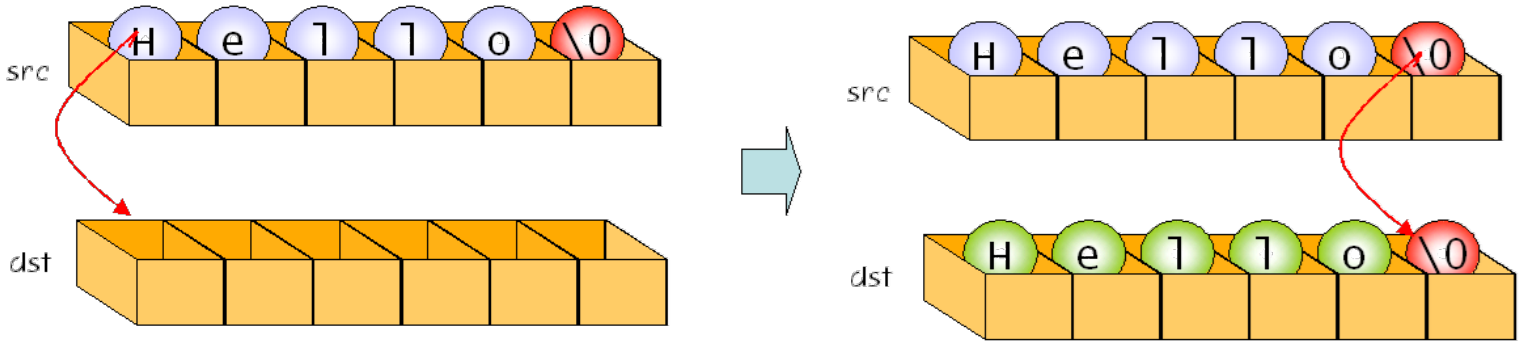
함수	설명
strlen(s)	문자열 s의 길이를 구한다.
strcpy(s1, s2)	s2를 s1에 복사한다.
strcat(s1, s2)	s2를 s1의 끝에 붙여넣는다.
strcmp(s1, s2)	s1과 s2를 비교한다.
strncpy(s1, s2, n)	s2의 최대 n개의 문자를 s1에 복사한다.
strncat(s1, s2, n)	s2의 최대 n개의 문자를 s1의 끝에 붙여넣는다.
strncmp(s1, s2, n)	최대 n개의 문자까지 s1과 s2를 비교한다.
strchr(s, c)	문자열 s안에서 문자 c를 찾는다.
strstr(s1, s2)	문자열 s1에서 문자열 s2를 찾는다.



문자열 길이, 복사

- 문자열의 길이
  - strlen(“Hello”)는 5를 반환
- 문자열 복사

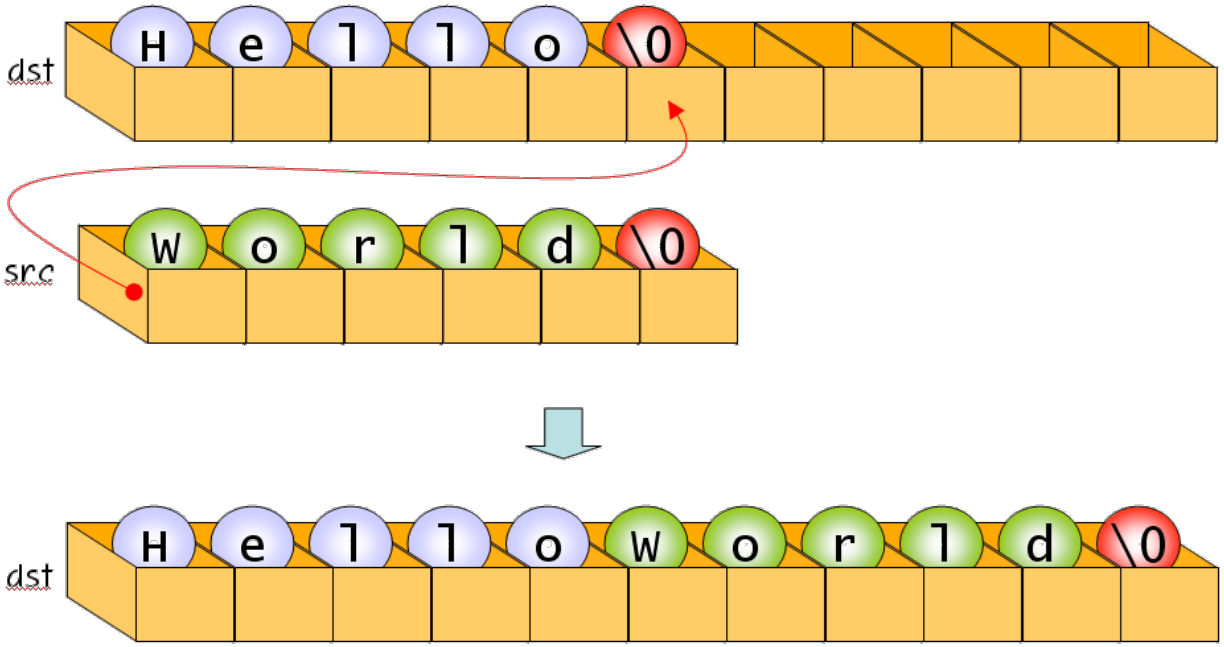
```
char dst[6];
char src[6] = “Hello";
strcpy(dst, src);
```



문자열 연결

- 문자열 연결

```
char dst[12] = "Hello";
char src[6] = "World";
strcat(dst, src);
```



예제

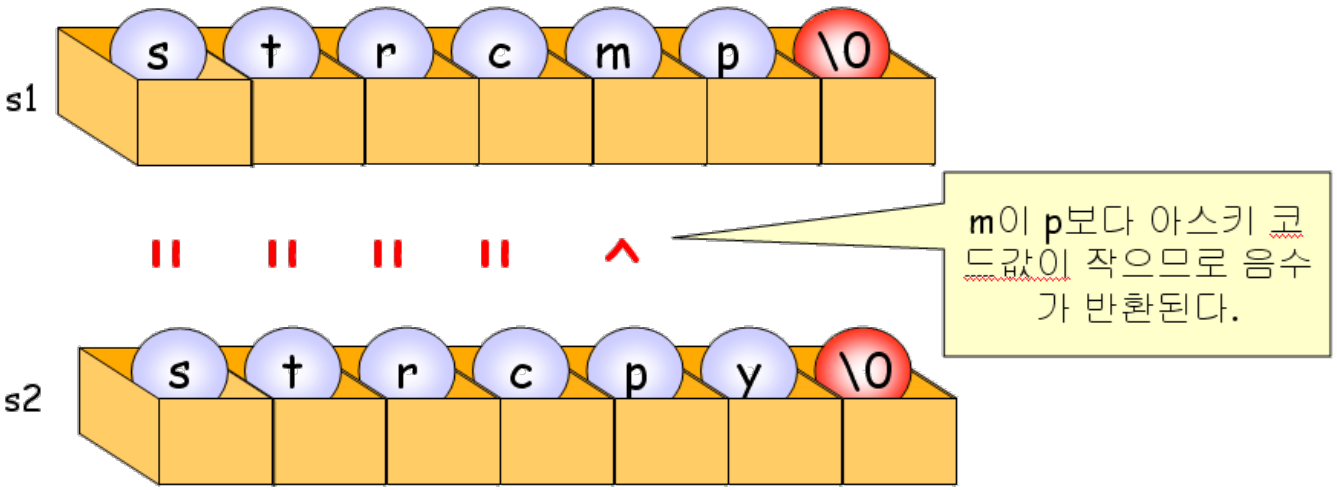
```
// strcpy와 strcat
#include <iostream>
#include <cstring>
using namespace std;

int main()
{
	char string[80];

	strcpy( string, "Hello World from " );
	strcat( string, "strcpy() " );
	strcat( string, "and " );
	strcat( string, "strcat()!" );
	cout << string << endl;
	return 0;
}
```

```
Hello World from strcpy() and strcat()!
```

문자열 비교

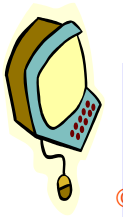
```
int strcmp( const char *s1,  const char *s2 );
```

반환값	s1과 s2의 관계
<0	s1이 s2보다 작다
0	s1이 s2와 같다.
>0	s1이 s2보다 크다.



예제

```
#include <iostream>
#include <cstring>
using namespace std;
int main()
{
	char s1[80];		// 첫번째 단어를 저장할 문자배열
	char s2[80];		// 두번째 단어를 저장할 문자배열
	int result;

	cout << "첫번째 단어를 입력하시오:";
	cin >> s1;
	cout << "두번째 단어를 입력하시오:";
	cin >> s2;
	result = strcmp(s1, s2);
	if( result < 0 )
		cout << s1 << "이 " << s2 << "보다 앞에 있습니다." << endl;
	else if( result == 0 )
		cout << s1 << "이 " << s2 << "와 같습니다." << endl;
	else
		cout << s1 << "이 " << s2 << "보다 뒤에 있습니다." << endl;
	return 0;
}
```



```
첫번째 단어를 입력하시오:cat
두번째 단어를 입력하시오:dog
cat이 dog보다 앞에 있습니다.
```

중간 점검 문제

1. C-문자열 src를 C-문자열 dst로 복사하는 문장을 써라.
2. “String"을 저장하려면 최소한 어떤 크기 이상의 문자 배열이 필요한가?
3. 문자열을 서로 비교하는 함수는?

Q & A