

C언어 강의자료

문정욱

A decorative graphic consisting of several light blue squares of varying sizes arranged in a stepped pattern on the left side of the slide.

C언어 더 알아보기 3

연산자

C언어 연산자 종류

종류	연산자
산술 연산자	+ - * / %
관계 연산자	> < == != >= <=
논리 연산자	! &&
증감 연산자	++ --
비트 연산자	& ^ ~ << >>

산술 연산자

산술 연산자(arithmetic operator)

우선순위	연산	연산자	피연산자 자료형	결과 값
1순위	단항 플러스	$+a$	정수형, 부동소수형	a의 값 그대로
	단항 마이너스	$-a$	정수형, 부동소수형	a의 부호를 변경한 값
2순위	곱셈	$a * b$	정수형, 부동소수형	a와 b의 곱
	나눗셈	a / b	정수형	a를 b로 나눈 몫
			부동소수형	a를 b로 나눈 값
	나머지	$a \% b$	정수형	a를 b로 나눈 나머지
3순위	덧셈	$a + b$	정수형, 부동소수형	a와 b의 합
	뺄셈	$a - b$	정수형, 부동소수형	a에서 b를 뺀 값

산술 연산자

산술 연산자의 피연산자와 결과값

연산자	정수형 피연산자	결과 값	실수형 피연산자	결과 값
$+a$	$+ \text{정수}$	정수	$+ \text{실수}$	실수
$-a$	$- \text{정수}$	정수	$- \text{실수}$	실수
$a * b$	정수 $*$ 정수	정수	실수 $*$ 실수	실수
a / b	정수 $/$ 정수	정수	실수 $/$ 실수	실수
$a \% b$	정수 $\%$ 정수	정수		
$a + b$	정수 $+$ 정수	정수	실수 $+$ 실수	실수
$a - b$	정수 $-$ 정수	정수	실수 $-$ 실수	실수

관계연산자

관계 연산자(relational operator)

연산	연산자	피 연산자 자료형	결과 값
같다	<code>a == b</code>	정수형, 부동소수형	<code>a</code> 값이 <code>b</code> 값과 같으면 1(참) 그렇지 않으면 0(거짓)
다르다	<code>a != b</code>	정수형, 부동소수형	<code>a</code> 값이 <code>b</code> 값과 같지 않으면 1(참) 그렇지 않으면 0(거짓)
작다	<code>a < b</code>	정수형, 부동소수형	<code>a</code> 값이 <code>b</code> 값보다 작으면 1(참) 그렇지 않으면 0(거짓)
작거나 같다	<code>a <= b</code>	정수형, 부동소수형	<code>a</code> 값이 <code>b</code> 값보다 작거나 같으면 1(참) 그렇지 않으면 0(거짓)
크다	<code>a > b</code>	정수형, 부동소수형	<code>a</code> 값이 <code>b</code> 값보다 크면 1(참) 그렇지 않으면 0(거짓)
크거나 같다	<code>a >= b</code>	정수형, 부동소수형	<code>a</code> 값이 <code>b</code> 값보다 크거나 같으면 1(참) 그렇지 않으면 0(거짓)

관계 연산자

관계 연산자의 피연산자와 결과값

연산자	정수형 피연산자	결과 값	실수형 피연산자	결과 값
<code>a == b</code>	정수 == 정수	진리값(정수)	실수 == 실수	진리값(정수)
<code>a != b</code>	정수 != 정수	진리값(정수)	실수 != 실수	진리값(정수)
<code>a < b</code>	정수 < 정수	진리값(정수)	실수 < 실수	진리값(정수)
<code>a <= b</code>	정수 <= 정수	진리값(정수)	실수 <= 실수	진리값(정수)
<code>a > b</code>	정수 > 정수	진리값(정수)	실수 > 실수	진리값(정수)
<code>a >= b</code>	정수 >= 정수	진리값(정수)	실수 >= 실수	진리값(정수)

논리 연산자

논리 연산자(logical operator)

연산	연산자	피연산자 자료형	결과 값
논리 NOT	<code>! a</code>	정수형	a 가 거짓이면 1(참) 이고, 그렇지 않으면 0(거짓) 이다.
논리 AND	<code>a && b</code>	정수형	a, b 모두 참이면 1(참) 이고, 그렇지 않으면 0(거짓) 이다.
논리 OR	<code>a b</code>	정수형	a, b 중 한 쪽이라도 참이면 1(참) 이고, 그렇지 않으면 0(거짓) 이다.

논리 연산자

논리 연산자의 피연산자와 결과값

연산자	진리값(정수형) 피연산자	결과 값
<code>! a</code>	<code>! 진리값(정수)</code>	진리값(정수)
<code>a && b</code>	<code>진리값(정수) && 진리값(정수)</code>	진리값(정수)
<code>a b</code>	<code>진리값(정수) 진리값(정수)</code>	진리값(정수)

명시적 타입 변환

- 타입 변환 연산자(type casting operator)
 - 강제적으로 값의 타입을 변경한다. → 값의 표현방식과 크기가 변경

입출력 결과

4

계속하려면 아무 키나 누르십시오 . . .

```
#include <stdio.h>

int main(void)
{
    char c=1;
    int i=2;
    float f=1.0f;
    double d=2.0f;

    i = i + (int)c ;
    f = f + (float)i ;
    d = d + (double)f ;
    i = i + (int)d ; // truncation

    printf("%d\n", sizeof( i+(int)d ) );
    return 0;
}
```

묵시적 타입 변환

- 묵시적 타입 변환(implicit type conversion) 원칙
 - 제1원칙: 정수형 \rightarrow 실수형
 - 제2원칙: char, short \rightarrow int (정수형에만 해당)
 - 제3원칙: 작은 크기 \rightarrow 큰 크기
 - char < short < int < long < long long
 - float < double < long double
 - 제4원칙: signed \rightarrow unsigned (정수형에만 해당)

묵시적 타입 변환

■ 제1원칙

정수 → 실수

```
#include <stdio.h>

int main(void)
{
    int      i=1;
    long     l=2;
    long long ll=3;
    float     f=1.0f;

    f = f + i ;    // f + (float)i
    f = f + l ;    // f + (float)l
    f = f + ll ;   // f + (float)ll
    return 0;
}
```

묵시적 타입 변환

■ 제2원칙

char → int
short → int

```
#include <stdio.h>

int main(void)
{
    char    c=1;
    short   s=2;
    int     i=3;

    i = c + c ;    // (int)c + (int)c
    i = c + s ;    // (int)c + (int)s
    i = s + s ;    // (int)s + (int)s
    return 0;
}
```

묵시적 타입 변환

■ 제3원칙

작은 타입 → 큰 타입

char → short → int → long → long long

float → double → long double

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    int        i=1;
```

```
    long       l=2;
```

```
    long long  ll=3;
```

```
    float      f=1.0f;
```

```
    double     d=2.0f;
```

```
    long double ld=3.0f;
```

```
    l = i + l ; // (long)i + l
```

```
    ll = i + ll ; // (long long)i + ll
```

```
    ll = l + ll ; // (long long)l + ll
```

```
    d = f + d ; // (double)f + d
```

```
    ld = f + ld ; // (long double)f + ld
```

```
    ld = d + ld ; // (long double)d + ld
```

```
    return 0;
```

```
}
```

묵시적 타입 변환

■ 제4원칙

정수 → 양의 정수

```
#include <stdio.h>

int main(void)
{
    int            i=1;
    unsigned int    ui=2;
    long long       ll=3;
    unsigned long long ull=4;

    ui = i + ui ;
        // (unsigned)i + ui
    ull = ll + ull ;
        // (unsigned long long)ll + ull
    return 0;
}
```


빈 문장

- 빈 문장(null statement)
 - 아무 동작도 하지 않는 문장
 - 아무 기술도 없이 세미콜론으로 끝난다.

variable (i)	condition (i < s && ..)	statement (null)

```
#include <stdio.h>

int main(void)
{
    int a[6]={3,2,6,8,10,9};
    int s=6;
    int value;
    int i;

    value=8;
    for(i=0;i<s && !(a[i]==value);++i)
        ; // null statement
    printf("index == %d\n",i);

    return 0;
}
```

입출력 결과

```
index == 3
계속하려면 아무 키나 누르십시오 . . .
```

증감 연산자

■ 증감 연산자 사용의 주의 점

- 피 연산자는 반드시 변수. 피 연산자로 수식은 안됨.
- 증감 연산자가 들어간 수식은 값이 된다.

```
++ a;
a = a + 1;
```

이 수식들의 결과는 값이 된다.

• 권고 사항

- 한 수식에서 한 변수에 대한 증감 연산자를 두 번 이상 사용하지 말 것. → 복잡해 짐

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    int a=2,b=1;
```

```
    a++++b; // syntax error
```

```
    ((a++)++) + b; // syntax error
```

```
    return 0;
```

```
}
```

이 수식의 결과는 값이 된다.

값은 ++연산자의 피연산자가 될 수 없다.

조건 연산자

■ 조건 연산자

- expr_1과 expr2는 타입이 같아야 한다.
- 만일 타입이 다를 경우 implicit type conversion이 발생한다.

conditional operator:

cond_expr ? expr_1 : expr_2

입출력 결과

2

8

계속하려면 아무 키나 누르십시오 . . .

```
#include <stdio.h>

int main(void)
{
    int a=1,b=2;
    int c,d;

    c = a>b ? a : b;
    printf("%d\n",c);

    d = sizeof(a<b ? 1 : 3.14);
    // a<b ? (double)1: 3.14
    printf("%d\n",d);

    return 0;
}
```

쉼표 연산자

- 쉼표 연산자(comma operator)
 - 수식의 값은 `expr_2`의 대표 값과 동일하다.
 - 일반적으로 대표 값의 활용은 권고하지 않는다.

comma operator:
`expr_1 , expr_2`

입출력 결과

```
1
2
계속하려면 아무 키나 누르십시오 . . .
```

```
#include <stdio.h>

int main(void)
{
    int a=1,b=2;
    int c;

    c = a, b ;    // (c=a), b;
    printf("%d\n", c );

    c = (a, b) ;    // c = (a,b);
    printf("%d\n", c );

    return 0;
}
```

쉼표 연산자

■ 일반적인 활용

- 두 개 이상의 수식을 한 문장에서 수행할 때 사용한다.
- 보통 for문의 초기화 및 증감 영역에서 주로 사용. 그 외 사용은 바람직하지 않다.

입출력 결과

```
0 4
1 3
2 2
3 1
4 0
계속하려면 아무 키나 누르십시오 . . .
```

```
#include <stdio.h>

int main(void)
{
    int a,b;

    for(a=0, b=4; a<5; ++a, --b)
        printf("%d %d\n", a, b);

    return 0;
}
```

쉘표 연산자

권고 코드

```
#include <stdio.h>

int main(void)
{
    int c;

    while( (c=getchar())!=EOF )
        putchar(c);

    return 0;
}
```

입출력 결과

```
This is a program to copy data.
This is a program to copy data.
Type Ctrl-Z to finish input.
Type Ctrl-Z to finish input.
^Z
계속하려면 아무 키나 누르십시오 . . .
```

비 권고 코드(수식 값 활용)

```
#include <stdio.h>

int main(void)
{
    int c;

    while( c=getchar(), c!=EOF )
        putchar(c);

    return 0;
}
```

입출력 결과

```
This is a program to copy data.
This is a program to copy data.
Type Ctrl-Z to finish input.
Type Ctrl-Z to finish input.
^Z
계속하려면 아무 키나 누르십시오 . . .
```

동일

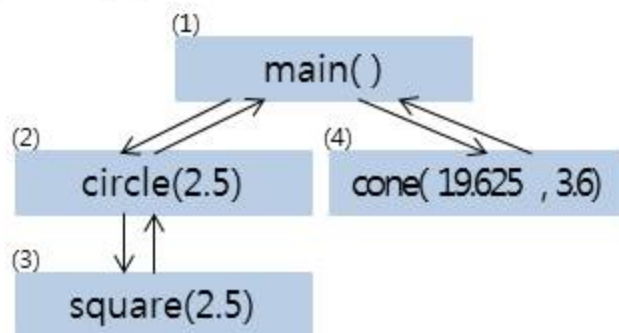
연산자

연산자 우선 순위

종류	연산자
구조체 연산자	-> .
단일 연산자	* & (type) sizeof ! ++ -- + -
산술 연산자	* / %
	+ -
관계 연산자	< <= > >=
	== !=
논리 연산자	&&
조건 연산자	? :
대입 연산자	= += -= *= /= %=
콤마 연산자	, (comma operator)

함수

■ 함수 호출시 지역 변수 생성과 소멸



RAM		
...	...	
4746	6.25	sq
4750	2.5	x
...	...	
4792	23.55	volume
4796	3.6	height
4800	19.626	base
...	...	
4968	23.55	cone_vol
...	...	

```

#include <stdio.h>

double square(double x)
{
    double sq;

    sq = x * x;
    return sq;
}

double circle(double r)
{
    double area;

    area = 3.14 * square(r);
    return area;
}

double cone(double base, double height)
{
    double volume;

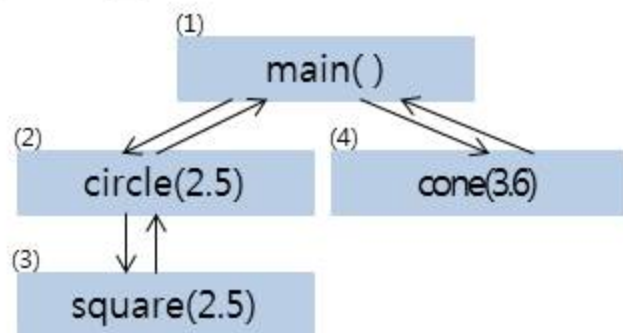
    volume = base * height / 3.0;
    return volume;
}

int main(void)
{
    double cone_vol;

    cone_vol = cone( circle( 2.5 ), 3.6 );
    return 0;
}
  
```

함수

■ 함수 호출시 전역 변수 생성과 소멸



RAM

...	...	
4750	2.5	x
...	...	
4800	3.6	height
...	...	
4968	23.55	cone_vol
...	...	
4792	23.55	g_var
...	...	

```

#include <stdio.h>

double g_var;

void square(double x)
{
    g_var = x * x;
}

void circle(double r)
{
    square(r);
    g_var = 3.14 * g_var;
}

void cone(double height)
{
    g_var = g_var * height / 3.0;
}

int main(void)
{
    double cone_vol;

    circle( 2.5 );
    cone( 3.6 );
    cone_vol = g_var;
    return 0;
}
  
```

변수

■ 전역 변수의 활용 (1)


• 이전 작업 정보의 기억

- 함수의 지역 변수는 함수가 끝나면 사라진다. 하지만, 전역변수를 사용하면 함수가 끝나도 정보를 저장할 수 있다.
- 그러므로 함수가 끝날 때 전역변수에 지금까지의 작업 내용을 저장해 두었다가 다시 함수가 호출될 때 전역변수를 통해 이전 작업 내용을 활용할 수 있다.

```
#include <stdio.h>
```

```
int s = 0;
```

```
int sum(int n)
```

```
{  
     이전 s 값에 n 값을 누적함  
    s += n;  
    return s;  
}
```

```
int main(void)
```

```
{  
    sum(1); // s += 1  
    sum(2); // s += 2  
    sum(3); // s += 3  
    printf("sum = %d\n", sum(0) );  
    return 0;  
}
```

변수

■ 전역 변수의 활용 (2)

• 함수 간의 정보 공유

- 함수가 끝날 때 전역변수에 다른 함수에게 전달할 정보를 저장해두었다가 다른 함수가 호출될 때 전역변수를 통해 이전 함수의 작업 내용을 활용할 수 있다.

※ 주의

전역 변수를 사용한 함수간 정보 공유는 함수의 구조를 간단하게 만들고 실행 속도를 높일 수 있다.

그러나 프로그램 전체 구조를 이해하기 위해서는 전역 변수와 함수와의 밀접한 관계를 반드시 이해해야 하므로 프로그램의 유지 보수를 어렵게 만들 수 있다.

그러므로 반드시 필요한 경우를 제외하면 전역변수를 사용한 함수간 정보 공유는 추천하지 않는다.

```
#include <stdio.h>
```

```
int s = 0;
```

```
int sum(int n)
```

```
{  
    s += n;  
    return s;  
}
```

전역 변수 *s*에
저장된 값을 공유함

```
void print_sum(void)
```

```
{  
    printf("sum = %d\n", s );  
}
```

```
int main(void)
```

```
{  
    sum(1);  
    sum(2);  
    sum(3);  
    print_sum();  
    return 0;  
}
```


변수

■ 정적(static) 지역 변수

- 실제는 전역 변수이지만 지역 변수의 특성을 가지고 있는 변수
- 가짜 지역 변수**
 - 전역 변수라고 가정하여 프로그램을 분석하면 된다.

변수 종류	생성 및 파괴	참조 범위
지역 변수	블록의 시작 및 종료	선언된 위치에서 블록의 끝까지
전역 변수	프로그램의 시작 및 종료	선언된 위치에서 프로그램의 끝까지
정적(static) 지역 변수	프로그램의 시작 및 종료	선언된 위치에서 블록의 끝까지

생성 및 파괴 규칙이 전역변수와 같으므로
이전 작업 정보의 기억에 활용은 가능

참조 범위가 지역변수와 같으므로
함수간의 정보 공유에 활용은 불가능

```
#include <stdio.h>
```

```
int s = 0;
```

변수 s는 외부영역에 선언되었다고 가정하여 분석해야 함

```
int sum(int n)
```

```
{
```

```
    static int s = 0;
```

```
    s += n;
```

```
    return s;
```

```
}
```

static 키워드가 없으면 s는 지역변수가 된다.
이전 작업 정보를 기억할 수 없다.

```
void print_sum( int v )
```

```
{
```

```
    printf("sum = %d\n", v );
```

```
}
```

```
int main(void)
```

```
{
```

```
    sum(1);
```

```
    sum(2);
```

```
    sum(3);
```

```
    print_sum( sum(0) );
```

```
    return 0;
```

```
}
```

함수

■ 값의 전달

• call by value

- 호출되는 함수에게 값을 전달하기 위해서는 인자(parameter)에 값을 복사하여 전달
- 인자(parameter)는 호출되는 함수의 내부 변수에는 영향을 주지 못함.

• return by value

- 호출하는 함수(calling)에게 값을 전달하기 위해서는 값을 반환(return)하여 전달

입출력 결과

```
a=4, b=6 in funct()
a=6, b=8 in funct()
c=5, d=6 in main()
e=4, f=6 in main()
계속하려면 아무 키나 누르십시오 . . .
```

```
#include <stdio.h>

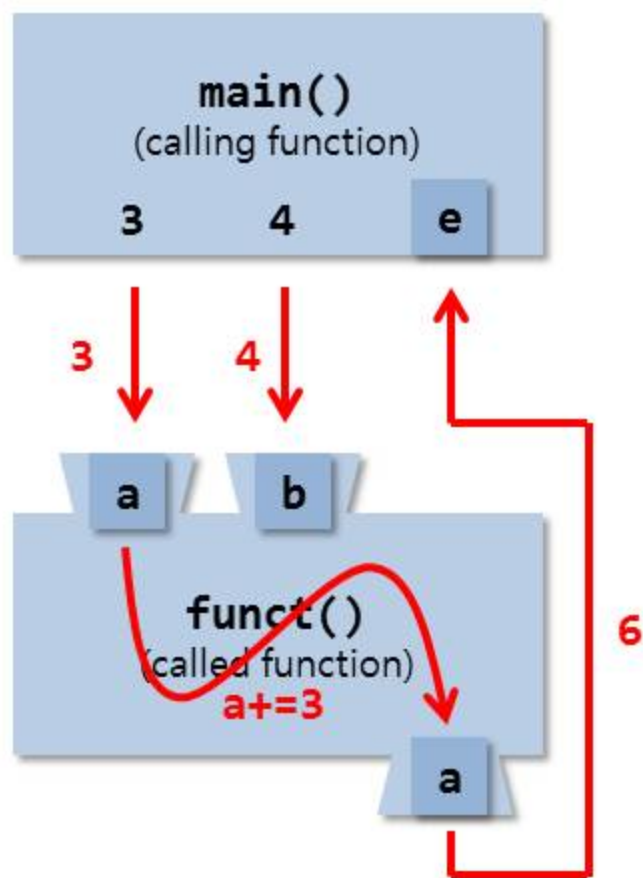
int funct(int a,int b)
{
    a+=1;
    b+=2;
    printf("a=%d, b=%d in funct()\n",a,b);
    return a;
}

int main(void)
{
    int c=5,d=6;
    int e,f;

    e = funct(3,4); // 3-->a, 4-->b
    f = funct(c,d); // c-->a, d-->b
    printf("c=%d, d=%d in main()\n",c,d);
    printf("e=%d, f=%d in main()\n",e,f);
    return 0;
}
```

함수

■ 상수 값의 전달



```
#include <stdio.h>

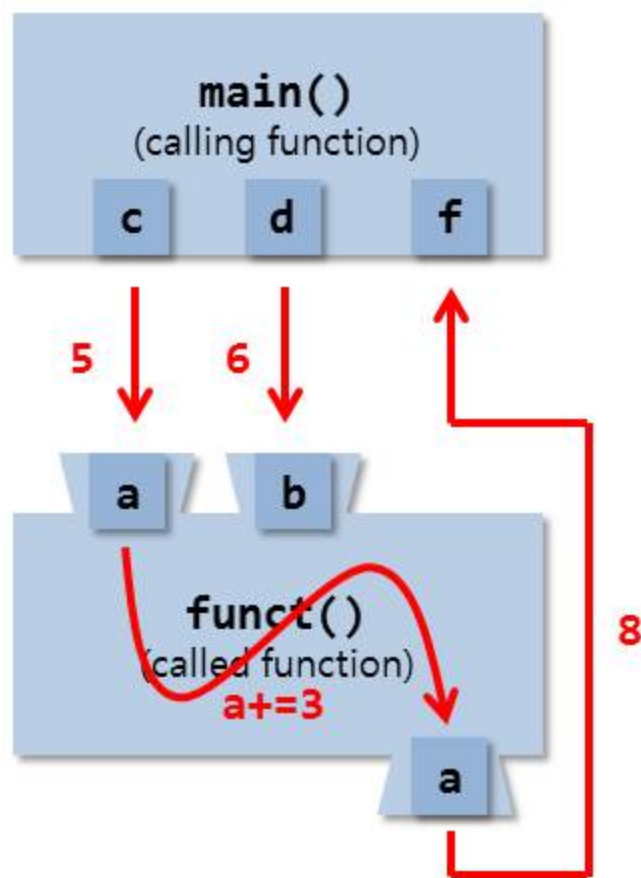
int funct(int a,int b)
{
    a+=3;
    b+=3;
    printf("a=%d, b=%d in funct()\n",a,b);
    return a;
}

int main(void)
{
    int c=5,d=6;
    int e,f;

    e = funct(3,4); // 3-->a, 4-->b
    f = funct(c,d); // c-->a, d-->b
    printf("c=%d, d=%d in main()\n",c,d);
    printf("e=%d, f=%d in main()\n",e,f);
    return 0;
}
```


함수

■ 변수 값의 전달



```
#include <stdio.h>

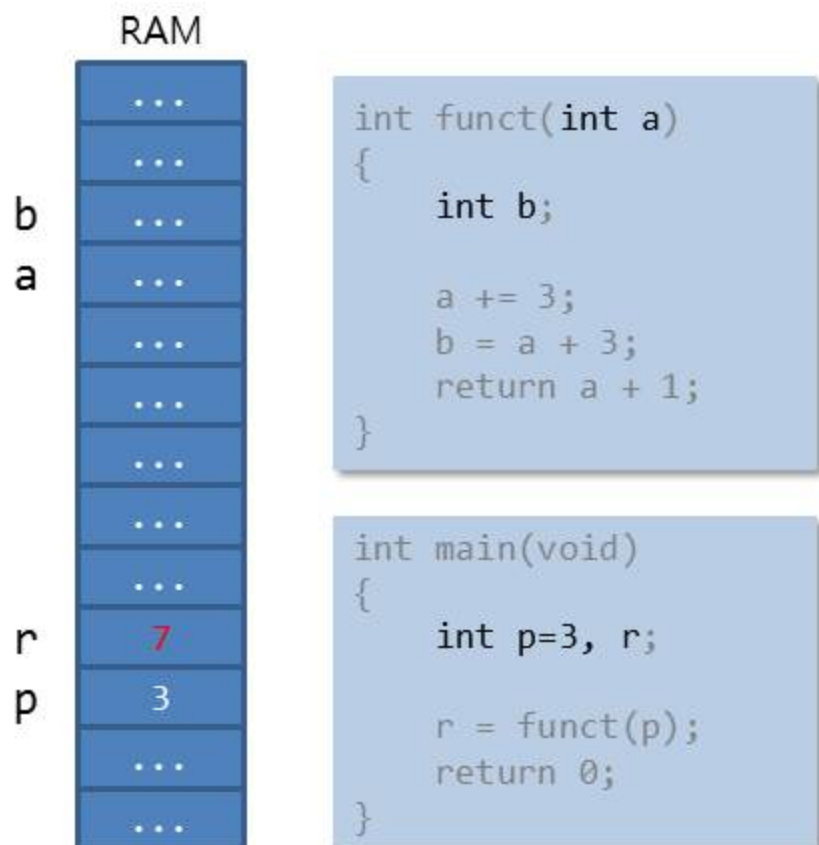
int func(int a,int b)
{
    a+=3;
    b+=3;
    printf("a=%d, b=%d in func()\n",a,b);
    return a;
}

int main(void)
{
    int c=5,d=6;
    int e,f;

    e = func(3,4); // 3-->a, 4-->b
    f = func(c,d); // c-->a, d-->b
    printf("c=%d, d=%d in main()\n",c,d);
    printf("e=%d, f=%d in main()\n",e,f);
    return 0;
}
```

함수

■ 값의 전달(call by value)



```
#include <stdio.h>

int func(int a)
{
    int b;

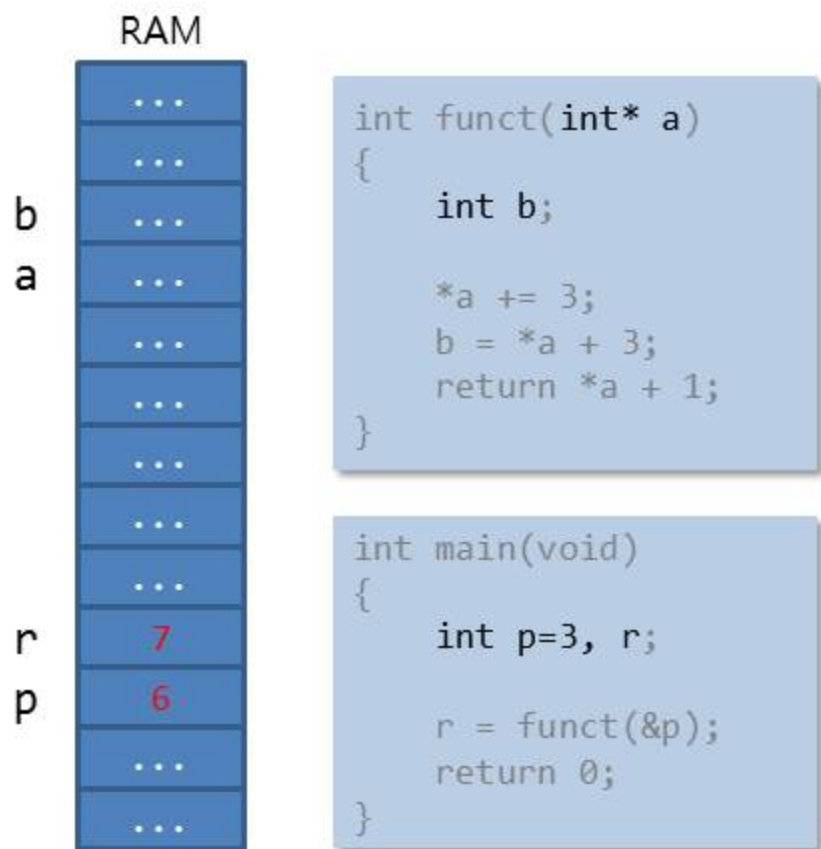
    a += 3;
    b = a + 3;
    return a + 1;
}

int main(void)
{
    int p=3, r;

    r = func(p);
    return 0;
}
```

함수

- 주소의 전달(call by address)



```
#include <stdio.h>

int funct(int* a)
{
    int b;

    *a += 3;
    b = *a + 3;
    return *a + 1;
}

int main(void)
{
    int p=3, r;

    r = funct(&p);
    return 0;
}
```

함수

Call by Value

```
#include <stdio.h>

void funct(int b)
{
    b += 3; // not change the value of a
}

int main(void)
{
    int a = 2;

    funct(a); // call by value
    printf("%d\n", a);
    return 0;
}
```

입출력 결과

2

계속하려면 아무 키나 누르십시오 . . .

Call by Address

```
#include <stdio.h>

void funct(int* p)
{
    *p += 3; // change the value of a
}

int main(void)
{
    int a = 2;

    funct(&a); //call by address
    printf("%d\n", a);
    return 0;
}
```

입출력 결과

5

계속하려면 아무 키나 누르십시오 . . .

Prototype of Function

■ 함수의 선언(declaration)

- Scope
 - 선언된 위치에서 파일 끝까지
- 형식
 - 리턴 타입, 함수 이름, 인자(변수) 타입

■ 함수의 정의(definition)

- Scope
 - 정의된 위치에서 파일 끝까지
- 형식
 - 리턴 타입, 함수 이름, 인자(변수) 타입, 인자 이름

```
#include <stdio.h>

// function prototype
int add(int,int);    // declaration

int main(void)
{
    int op1=1;
    int op2=2;

    printf("sum=%d\n", add(op1,op2) );
    return 0;
}

int add(int a,int b)    // definition
{
    return a+b;
}
```