전산 SMP 2주차

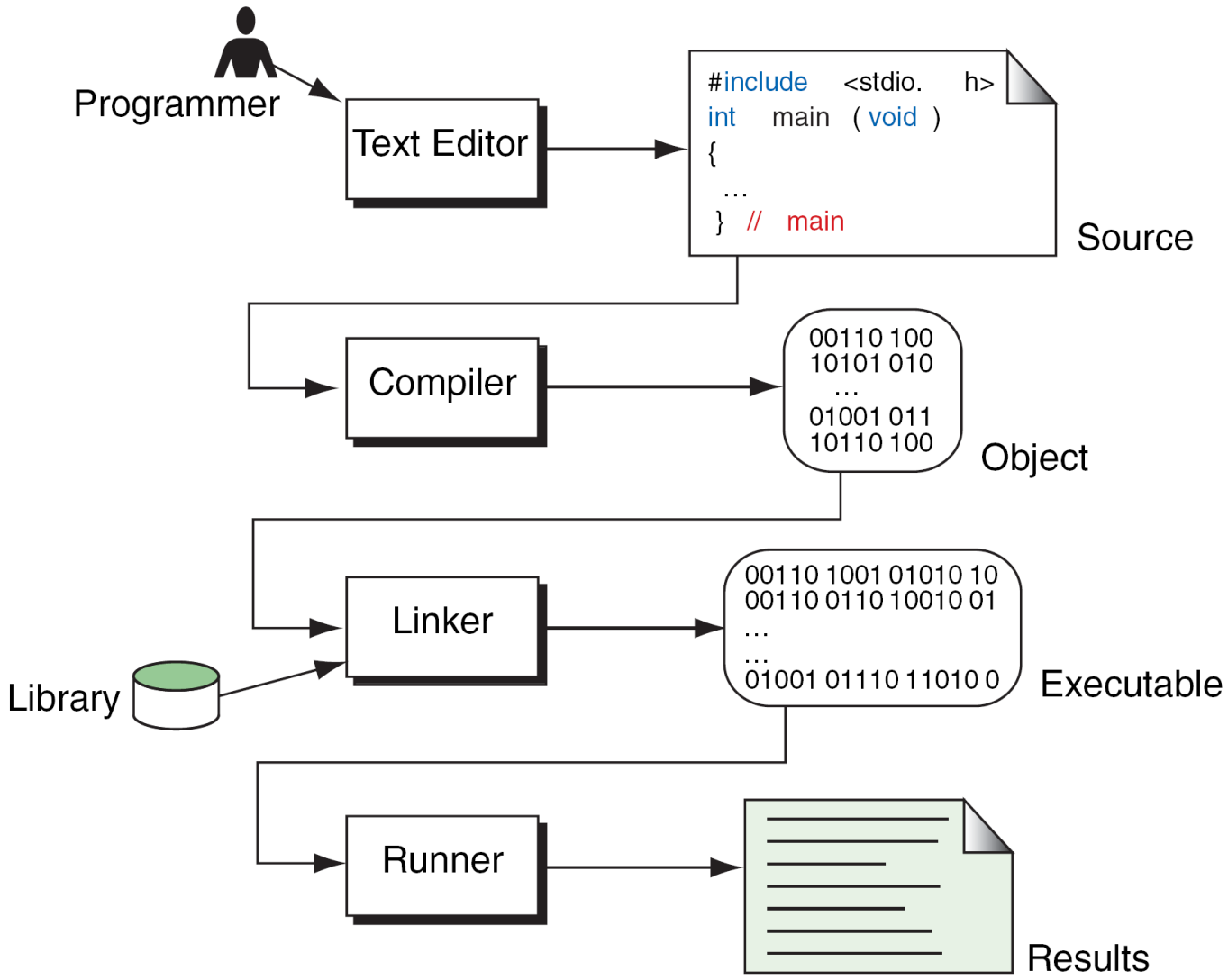
2014. 10. 05

김범수

bskim45@gmail.com

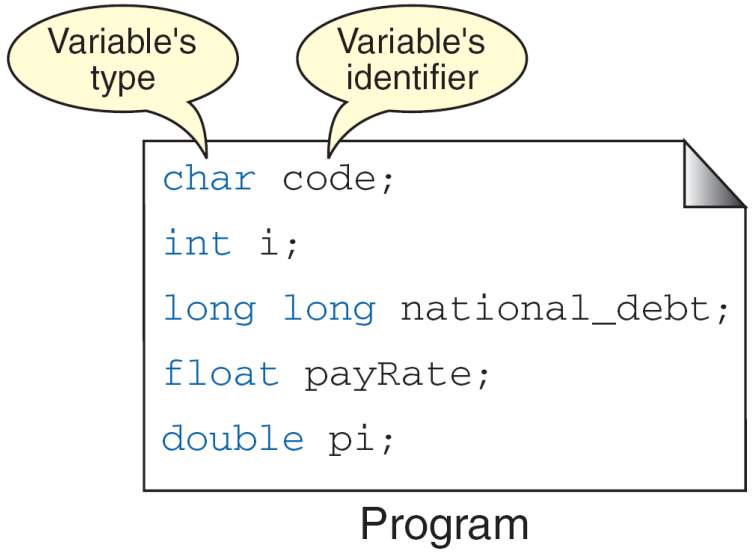
Special thanks to 박기석 (kisuk0521@gmail.com)

지난시간 복습


Programmer
Text Editor
#include <stdio. h>
int main (void)
{
…
} // main
Source
Compiler
00110 100
10101 010
…
01001 011
10110 100
Object
Linker
00110 1001 01010 10
00110 0110 10010 01
…
…
01001 01110 11010 0
Executable
Library
Runner
Results

변수란?

- 값을 저장할 수 있는 메모리공간에 붙여진 이름
- 변수라는것을 선언하면 메모리 공간이 할당되고 할당된 메모리 공간에 이름이 붙는다.
- 일반변수와 포인터 변수
- const키워드 사용 : 변수를 상수화시킨다.



int num; // int : 정수의 저장을 위한 메모리 공간의 할당
num : 할당된 메모리 공간의 이름은 num

num=20; //변수 num에 접근하여 20을 저장

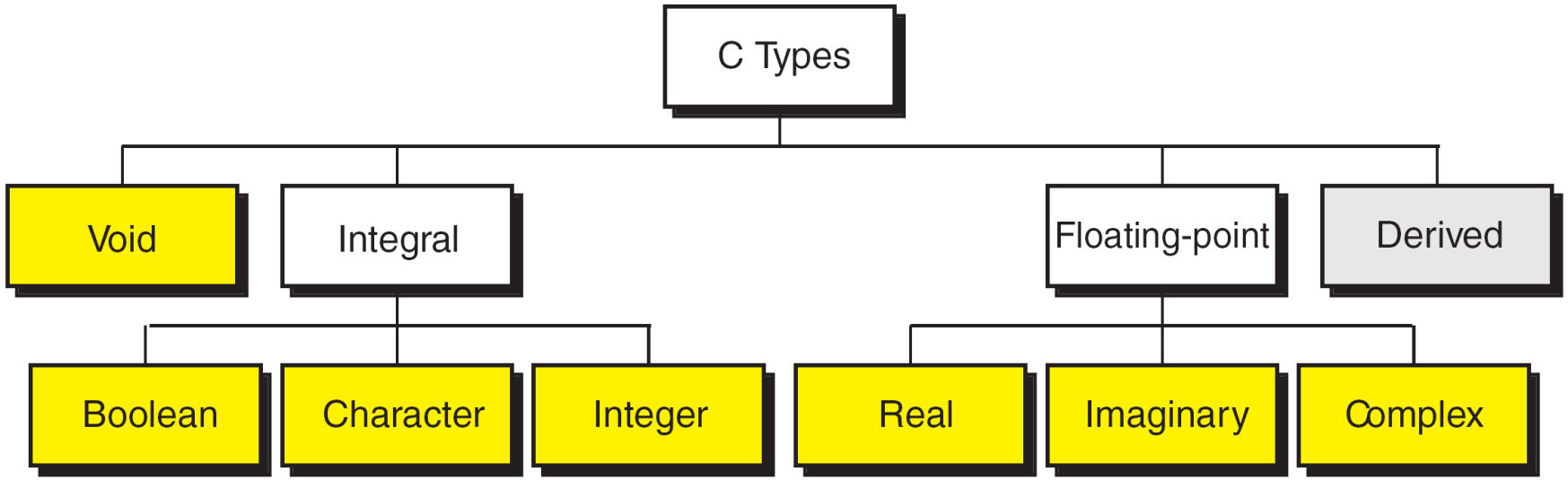
printf("%d", num); // num 변수안의 값을 정수형태로 출력

#define

- Preprocessor Directive 부분에 선언
- 컴파일 시 자동으로 샥샥 대체되어 들어간다.

```
#define PI 3.141592     //세미콜론 안 붙임
….
float pi = PI;          //float pi=3.141592;
```

자료형


C Types
Void
Integral
Floating-point
Derived
Boolean
Character
Integer
Real
Imaginary
Complex

기본 자료형의 종류와 데이터의 표현 범위

자료형		크기	값의 표현범위
정수형	char	1byte	$-128 \sim 127\ (-2^7 \sim 2^7-1)$
	unsigned char	1byte	$0 \sim 255\ (0 \sim 2^8-1)$
	short	2byte	$-2^{15} \sim 2^{15}-1$
	unsigned short	2byte	$0 \sim 2^{16}-1$
	int	4byte	$-2^{31} \sim 2^{31}-1$
	unsigned int	4byte	$0 \sim 2^{32}-1$
	long	4byte	$-2^{31} \sim 2^{31}-1$
	unsigned long	4byte	$0 \sim 2^{32}-1$
	long long	8byte	$-2^{63} \sim 2^{63}-1$
	unsigned long long	8byte	$0 \sim 2^{64}-1$
실수형	float	4byte	$\pm 3.4 \times 10^{-37} \sim \pm 3.4 \times 10^{+38}$
	double	8byte	$\pm 1.7 \times 10^{-307} \sim \pm 1.7 \times 10^{+308}$
	long double	8byte이상	double이상의 표현범위

형변환 (Type Conversion)

- Implicit Conversion
  - 컴파일러에 의해 사용자 모르게 자동으로 이루어 지는 형변환
  - Promotion, Demotion : 연산시 우선순위 높은 거에 맞춰서 다른 데이터 변환 (bool 〈 char 〈 integer 〈 real)
  - 최종적으로 값 대입시 대입 받는 변수의 타입 따라 감.
- Explicit Conversion(cast)
  - 프로그래머가 인위적으로 타입을 바꿈. 사용시 주의할것!

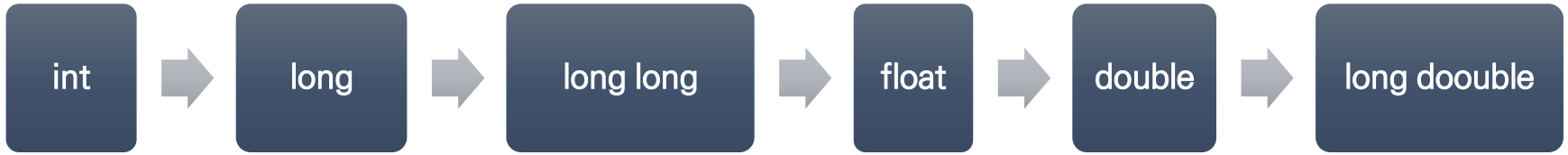
  double a = (double) 4/3

  double b = (double) 4 / (double) 3

  a=? b=?

자동 형 변환 (산술연산에서의 형 변환 규칙)

```
double a=1.7 + 30; // a=31.7
int b=1.7 + 30; // b=31

main()
{
        int a=-1;
        unsigned int b=100;
        if(a>b)
                printf("a가 크다");
        else
                printf("b가 크다");
}
```



printf

```
Int a = 1;
Int b = 2;
Int c = 3;

Printf("%d  %d  %d\n", a, b, c);
```

서식문자(Conversion Specifier) : 데이터의 출력 형태를 지정할 수 있다.
%d : 부호있는 10 진수로 출력하라!

scanf

```
int a;
char b;

scanf("%d %c", &a, &b);
```

Scanf로 값을 받기 위해서는 받는 위치, 즉 주소값을 인자로 주어야 한다.
& 가 변수 이름 앞에 붙을 시 그 변수가 가리키는 주소를 의미한다.

a : 변수 이름, 메모리 공간에 붙은 가상의 이름
&a : 변수 a 의 주소

서식문자 옵션은 precision, flag, size 값 줄 수 없다. 오직 width만 가능

scanf

- 첫번 째 인자의 서식문자와 뒤의 변수가 대응해야 한다.
  - 갯수, 타입
- 변수에 값을 넣기위해서는 변수의 주소! 를 주어야 한다.
  - &a
- 첫번째 인자(format string)에서 공백, 서식문자를 제외한 문자가 있으면 입력시 형태를 맞춰줘야 한다.
  - “%d-%d” -〉 1-2
- Format string 마지막에 공백이 있으면 안된다.!
  - “%d %d ” X -〉 “%d %d” O

printf()

```
#include <stdio.h>

int main(void)
{
   printf("%c %d %s \n", 'A', 'A', "ABC");
   printf("%10d \n",123); // 오른쪽정렬
   printf("%-10d \n",123); // 왼쪽 정렬
   printf("%10.2f \n",123.456789); // 왼쪽 정렬 ****123.46(*은 빈칸)
   printf("%+f %+f\n",123.45,-123,45); // +123.45 -123.45
   printf("% f % f\n", 123.45,-123,45); // *123.45 -123.45(*은 빈칸)
   printf("%-6s %6s \n", "AAA", "BBB");
}
// 소수점 아래는 6자리까지만 문제가 없고 , 7자리부터 문제 발생
```

scanf()

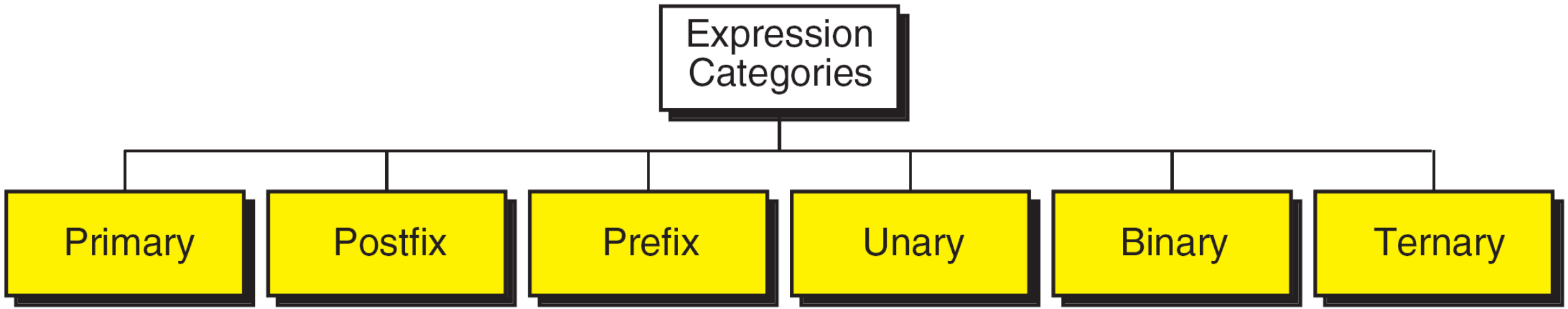
```
#include <stdio.h>

int main(void)
{
   int num1, num2, num3;
   printf("세 개의 정수 입력: ");
   scanf("%d %o %x", &num1, &num2, &num3); //12 12 12

   printf("입력된 정수 10진수 출력: ");
   printf("%d %d %d \n", num1, num2, num3); // 12 10 8
   return 0;
}
```

Assn #1 리뷰

1. 구의 반지름을 받아 부피와 겉넓이 구하기
   A. 구의 반지름 float
   B. #define PI
   C. 이름 ASCII code
   D. scanf로 입력 받기

2. 속도와 거리로 걸린시간 구하기
   1. double 실수 scanf 입력
   2. 결과는 소수점 둘째자리
   3. 걸린 시간은 O초 = O시간 O분 O초

연산자


Expression Categories
Primary
Postfix
Prefix
Unary
Binary
Ternary

Primary Expressions

- Names
  - a, b12, price, INT_MAX, SIZE
- Literal constants
  - 5, 123.98, ‘A’, “Welcome”
- Parenthetical expressions
  - (2 * 3 + 4), (a = 23 + b * 6)

Prefix/Postfix/Unary

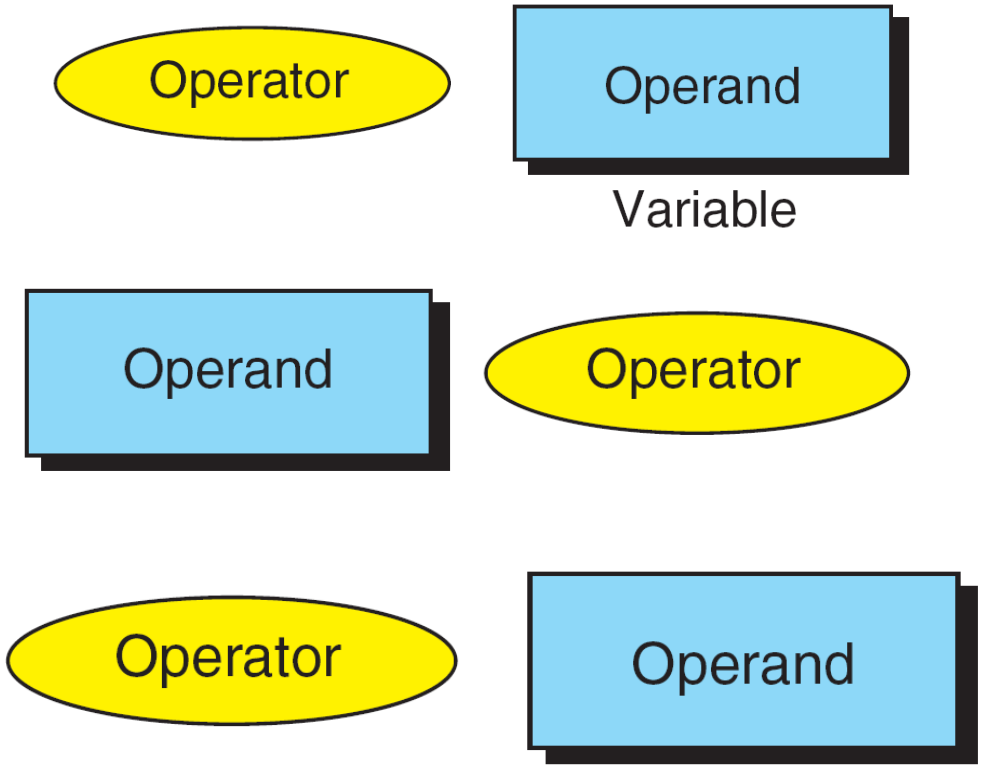
Prefix

- 증가/감소 연산자: --a, ++a

Postfix

- 증가/감소 연산자: ++a, --a
- 함수 호출

Unary



연산자의 종류

- 산술연산자 : +,−, *, /, %
- 대입연산자 : =, +=, −=, *=, /=, %=, 〈〈=, 〉〉=, &=, ^=, |=
- 관계(비교)연산자 : 〉, 〈, 〉=, 〈=, ,!=
- 논리연산자 : &&, ||, !
- 증가/감소 연산자 : ++, −−
- 비트연산자 : &, |, ^, ~
- 시프트 연산자 : 〈〈, 〉〉
- 주소 연산자 : &

산술연산자

종 류	사용 방법	설 명
+	a + b	두 개의 피연산자들을 덧셈
-	a - b	두 개의 피연산자들을 뺄셈
*	a * b	두 개의 피연산자들을 곱셈
/	a / b	변수 a를 변수 b로 나눗셈
%	a%b	변수 a를 변수 b로 나누어 나머지를 구함
-	- a	변수 a의 부호를 변경

```
int a=22, b=5;
printf("%d + %d = %d \n", a, b, a+b); // 22+5=27
printf("%d - %d = %d \n", a, b, a-b); // 22-5=17
printf("%d × %d = %d \n", a, b, a*b); // 22 x 5=110
printf("%d ÷ %d의 몫 = %d \n", a, b, a/b);  // 22÷5=4
printf("%d ÷ %d의 나머지 = %d \n", a, b, a%b);// 22÷5=2
```

복합 대입연산자

종 류	사용방법	설 명
a+=b	a=a+b	a+b 의 값을 a에 대입
a-=b	a=a-b	a-b 의 값을 a에 대입
a*=b	a=a*b	a*b 의 값을 a에 대입
a/=b	a=a/b	a/b 의 값(몫)을 a에 대입
a%=b	a=a%b	a%b 의 값(나머지)을 a에 대입

```
#include 〈stdio.h〉
int main(void) {
    int a=7, b=2,c=10;
    a+=3;
    b*=3;
    c%=3;
    printf(“결과 : %d ,%d , %d \n", a, b, c); // 결과 : 10, 6 , 1
}
```

비교(관계) 연산자

종 류	사용방법	설 명
〉	a 〉 b	변수 a가 변수 b보다 크면 참, 작거나 같으면 거짓
〈	a 〈 b	변수 a가 변수 b보다 작으면 참, 크거나 같으면 거짓
〉=	a 〉= b	변수 a가 변수 b보다 크거나 같으면 참, 작으면 거짓
〈=	a 〈= b	변수 a가 변수 b보다 작거나 같으면 참, 크면 거짓
==	a == b	변수 a와 변수 b가 같으면 참, 같지 않으면 거짓
!=	a != b	변수 a와 변수 b가 같지 않으면 참, 같으면 거짓

```
int a=10, b=12;
printf("result1: %d \n", a==b);  // result1 : 0
printf("result2: %d \n", a!=b);  // result2 : 1
printf("result3: %d \n", a〉=b);  // result3 : 0
return 0;
```

논리연산자

종 류	사용 방법	설 명
&&	a && b	a와 b가 모두 참인 경우에만 참, 둘 중 하나라도 거짓이면 거짓
||	a || b	a와 b중에 하나만 참이거나 둘 모두 참인 경우에는 참, 둘 다 거짓인 경우에는 거짓
!	!a	a가 참인 경우에는 거짓, a가 거짓인 경우에는 참

```
#include <stdio.h>
int main(void) {
    int a=10, b=12;
    int result1, result2, result3;

    result1 = (a==10 && b==12);
    result2 = (a<12 || b>12);          //  result2 =
    (a<(12 || b)>12);  은 결과 0
    result3 = (((!a)));
    printf("result1: %d \n", result1);  // 1
    printf("result2: %d \n", result2);  // 1
    printf("result3: %d \n", result3);  // 0
    return 0;
}
```

증가/감소 연산자

종 류	사용방법	설 명
++	++a	값을 1증가 후 나머지 연산 진행
	a++	++이외의 연산을 먼저 진행 후 값을 1증가
--	--a	값을 1감소 후 나머지 연산 진행
	a--	--이외의 연산을 먼저 진행 후 값을 1감소

```
#include <stdio.h>

int main(void) {
        int a=0, b=0;
        while(a<3){
                printf("a=%d  b=%d\n", a++, ++b);
}
```

증감 연산자 예제

```
3       Date:
4   */
5   #include <stdio.h>
6   int main (void)
7   {
8   // Local Declarations
9      int a;
10
11  // Statements
12     a = 4;
13     printf("value of a    : %2d\n", a);
14     printf("value of ++a  : %2d\n", ++a);
15     printf("new value of a: %2d\n", a);
16     return 0;
17  }  // main
```

```
Results:
value of a    :  4
value of ++a  :  5
new value of a:  5
```

```
5   #include <stdio.h>
6   int main (void)
7   {
8   // Local Declarations
9      int a;
10
11  // Statements
12     a = 4;
13     printf("value of a    : %2d\n", a);
14     printf("value of a++  : %2d\n",   a++);
15     printf("new value of a: %2d\n\n", a);
16     return 0;
17  }  // main
```

```
Results:
value of a    :  4
value of a++  :  4
new value of a:  5
```

비트 연산자

연산자	종류	사용방법	설 명
비트 논리 연산자	&	a&b	변수 a와 변수b의 비트들을 비교하여 각 비트가 모두 1인 경우에만 1, 그렇지 않으면 0
	|	a|b	변수 a와 변수b의 비트들을 비교하여 각 비트 중 하나라도 1인 경우에만 1, 그렇지 않으면 0
	^	a^b	변수 a와 변수 b의 비트가 다른 경우에는 1, 같으면 0
	~	~a	변수a의 비트가 1인 경우에는 0, 0인 경우는 1
시프트 연산자	〈〈	a〈〈3	a의 비트들을 왼쪽으로 3비트만큼 이동 1비트만큼 이동할 때마다 a*2만큼 증가
	〉〉	a〉〉3	a의 비트들을 오른쪽으로 3비트만큼 이동 1비트만큼 이동할 때마다 a/2만큼 감소

Shift 대상		공백 메우기
〈〈 (Left Shift)	signed	0으로 채워짐
	unsigned	
〉〉 (Right Shift)	signed	Sign bit의 값으로 채워짐
	unsigned	0으로 채워짐

비트 연산자–1

```
#include <stdio.h>

int main(void)
{
   int a = 14;         // 00000000 00000000 00000000 00001110
   int b = 20;         // 00000000 00000000 00000000 00010100
   int c = a & b;
   int d = a | b;
   int e = a^b;
   int f = ~a;
   printf("연산의 결과: %d %d %d \n", c, d, e, f);  // 4, 30, 26 , -15
}
```

비트 연산자–2

```
#include 〈stdio.h〉

int main(void)
{
    int a=15;      // 00000000 00000000 00000000 00001111
     int b=−15;      // 11111111 11111111 11111111 11110000
     int c=15;      // 11111111 11111111 11111111 11110000

    printf(“%d \n", a〈〈2);  // 60
    printf(“%d \n", b〉〉2);  // −4
    printf(“ %d \n", c〉〉2);  // 3
    return 0;
}
```

sizeof 연산자

- **sizeof : 변수 또는 자료형의 크기를 byte수로 알려준다.**

```
#include <stdio.h>
int main(void)
{
    char ch='A';
    int a=987;
    double b=3.1415;
    printf("변수 ch의 크기: %d \n", sizeof(ch));
    printf("변수 a의 크기: %d \n", sizeof(a));

    printf("int의 크기: %d \n", sizeof(int));
    printf("float의 크기: %d \n", sizeof(float));
    printf("double의 크기: %d \n", sizeof(double));

    printf("'A'의 크기: %d \n", sizeof('A'));
    printf("\"ABC\"의 크기: %d \n", sizeof("ABC"));
    printf("3의 크기: %d \n", sizeof(3));
    printf("1.234의 크기: %d \n", sizeof(1.234));
}
```

sizeof 연산자

```
#include <stdio.h>

int main(void)
{
    char c1=1, c2=2, result1=0;
    short s1=300, s2=400, result2=0;
    float f1=1.2, f2=3.4;
    printf("c1 & c2: %d, %d \n", sizeof(c1), sizeof(c2)); // 1,1
    printf("s1 & s2: %d, %d \n", sizeof(s1), sizeof(s2)); // 2,2
    printf("f1 & f2: %d, %d \n", sizeof(f1), sizeof(f2));  // 4,4

    printf("sizeof(c1+c2): %d \n", sizeof(c1+c2));  // 4
    printf("sizeof(c1)+sizeof(c2): %d \n", sizeof(c1)+sizeof(c2));
    // 2
    printf("sizeof(s1+s2): %d \n", sizeof(s1+s2));  // 4
    printf("sizeof(s1)+sizeof(s2): %d \n", sizeof(s1)+sizeof(s2)); //
    4
    printf("sizeof(f1+f2): %d \n", sizeof(f1+f2));  // 4
    printf("sizeof(f1)+sizeof(f2): %d \n", sizeof(f1)+sizeof(f2));  //
    8

    result1=c1+c2;
    result2=s1+s2;
    printf("result1 & result2: %d, %d \n", sizeof(result1),
    sizeof(result2)); // 1,2
}
```

삼항 연산자

- 조건식 ? 실행1 : 실행2

조건식이 참이면 실행1 문장을,

조건식이 거짓이면 실행 2문장을 수행

```
#include <stdio.h>

int main(void)
{
        int num=1, res;

        res = (num==1) ? ++num : --num;
        printf("결과: %d \n", res);  // 2
}
```

Try

- int a = 2;
- int b = 4;
- int num = 3;

- a *= (num−−) == 4 || a+2 〈= 5−1 && ++b 〉=3;

- a=? num=? b=?

연산자 우선순위와 결합순서

우선 순위	연산 기호	연산자	결합방향
1위	()	함수호출	→
	[]	인덱스	
	->	간접지정	
	.	직접지정	
2위	++	증가	←
	--	감소	
	sizeof	바이트 단위 크기 계산	
	~	비트단위 NOT	
	!	논리 NOT	
	-,+	부호 연산(음수, 양수)	
	&	주소 연산	
	*	간접 지정 연산	
3위	(casting)	자료형 변환	←
4위	*,/,%	곱셈, 나눗셈,나머지	→
5위	+,-	덧셈, 뺄셈	→
6위	<<,>>	비트이동	→
7위	<,>,<=,>=	대소비교	→
8위	==,=!	동등비교	→
9위	&	비트 AND	→
10위	^	비트 XOR	→
11위	|	비트 OR	→
12위	&&	논리 AND	→
13위	||	논리 OR	→
14위	? :	조건 연산	←
15위	=,+=,-=,*,/=,%=,<,<=,>>=,&=,^=,|=	대입 연산	←
16위	,	콤마연산	→

연산자 우선순위

- 기본적으로 왼쪽에서 오른쪽으로

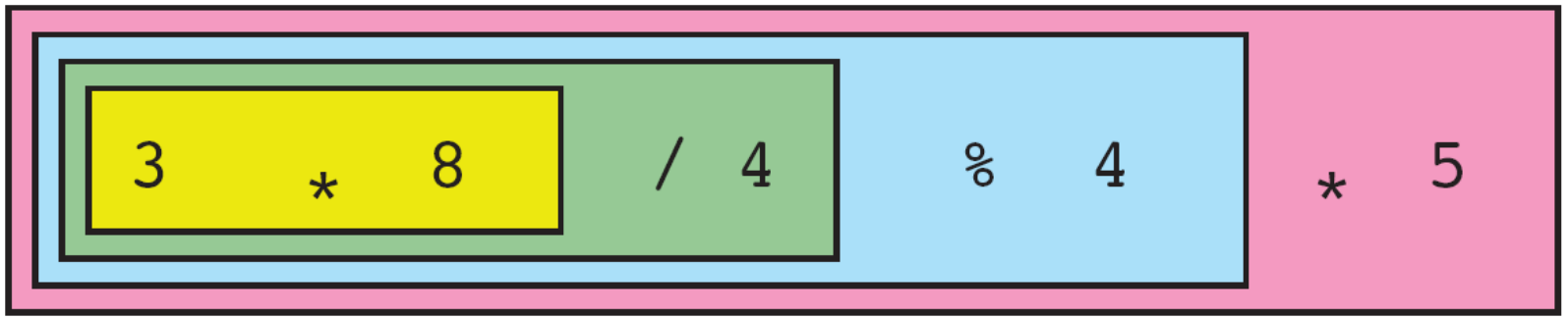
- A+B*C/D-E
  1. *
  2. /
  3. +
  4. -
- 단, 대입 연산자의 경우 오른쪽에서 왼쪽으로
- A = b = C+4
- +먼저 계산하고 b에 c+4를 먼저 대입하고, 그 다음에 A에 b를 대입합니다.

```c
#include <stdio.h>

int main (void)
{
// Local Declarations
   int a = 10;
   int b = 20;
   int c = 30;

// Statements
   printf ("a *  b + c  is: %d\n", a *  b + c);
   printf ("a * (b + c) is: %d\n", a * (b + c));
   return 0;
}  // main
```

```
Results:
a *  b + c  is: 230
a * (b + c) is: 500
```

연산자 결합순서

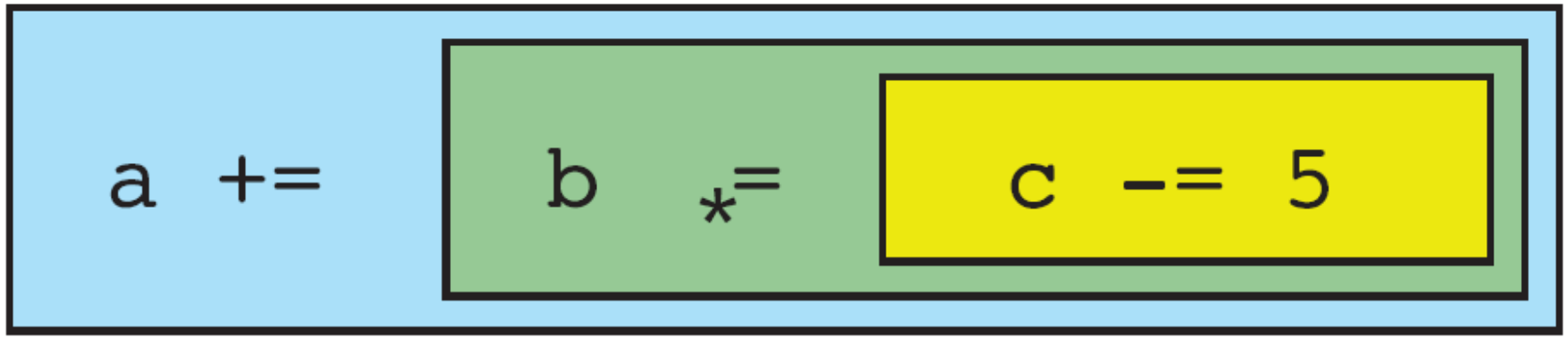
- a + b + c + d
- a * b * c * d



Left-to-Right Associativity

연산자 결합순서

- a = b = c = d = 10



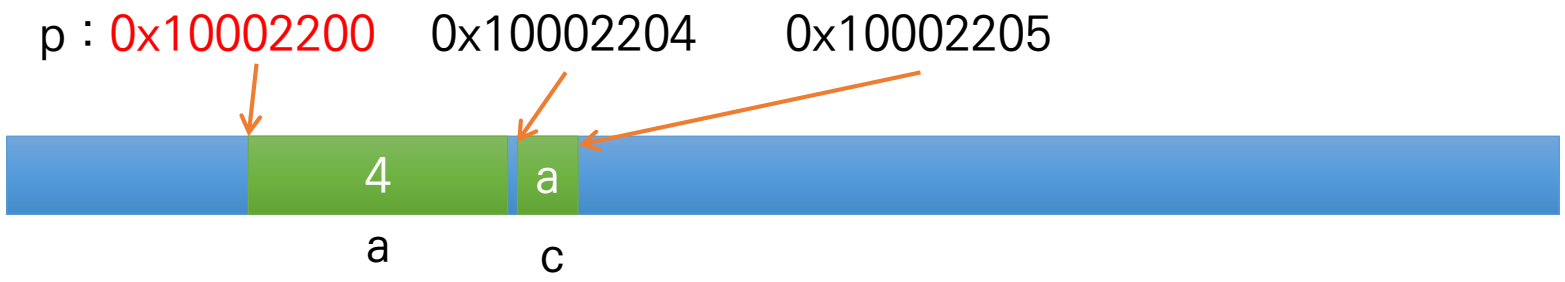
Right-to-Left Associativity

Pointer 맛보기

포인터의 개념

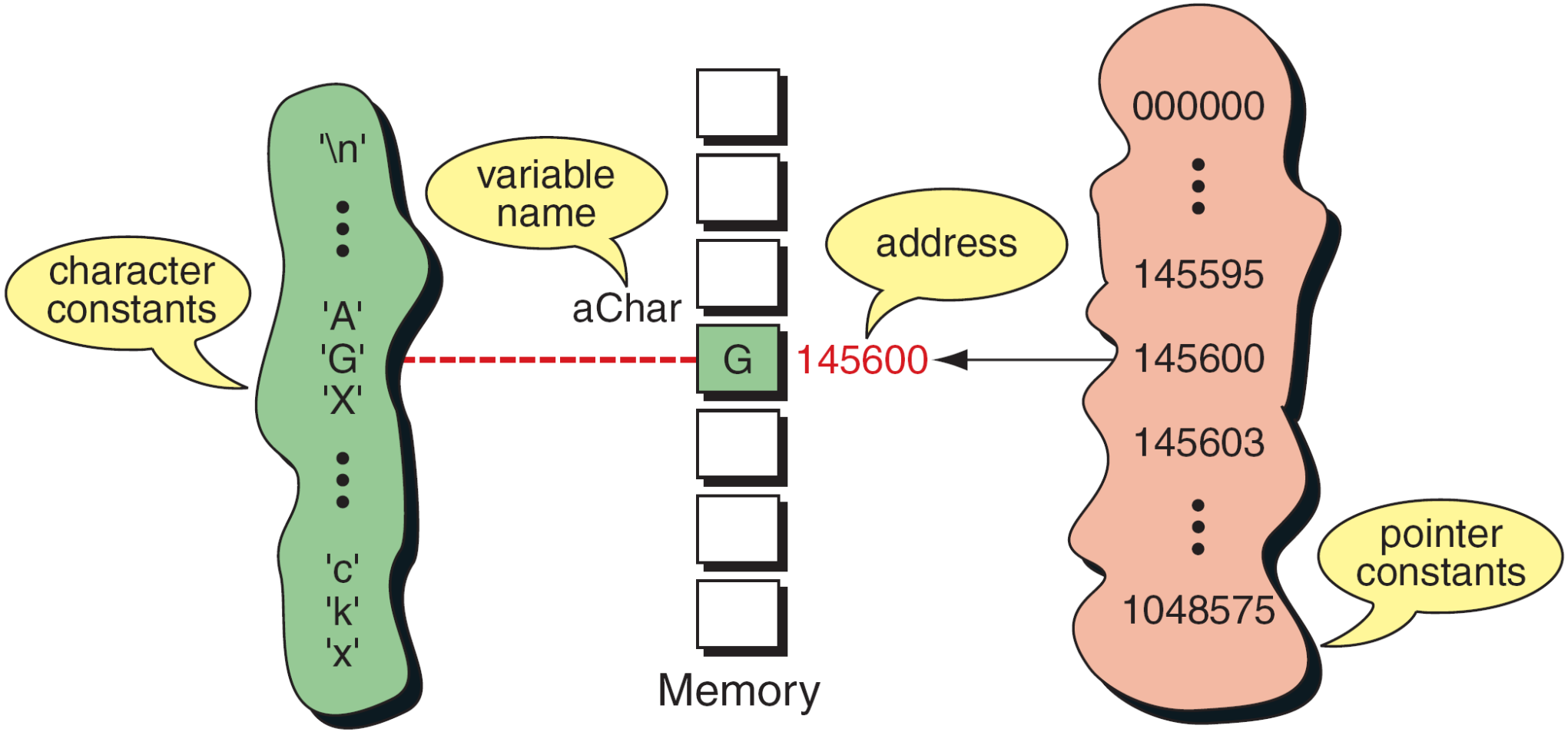
Pointer란?

- 데이터 접근에 사용되는 주소를 저장하는 타입 – 포인터도 타입이다
- int a = 4; char c = ‘a’; int *p = &a;



- 메모리는 긴 일차원 공간으로 볼 수 있고, 각 byte는 자신만의 주소를 가지고 있다.

변수와 주소와의 관계


'\n'
'A'
'G'
'X'
'c'
'k'
'x'
character constants
variable name
aChar
G
145600
address
000000
145595
145600
145603
1048575
pointer constants
Memory

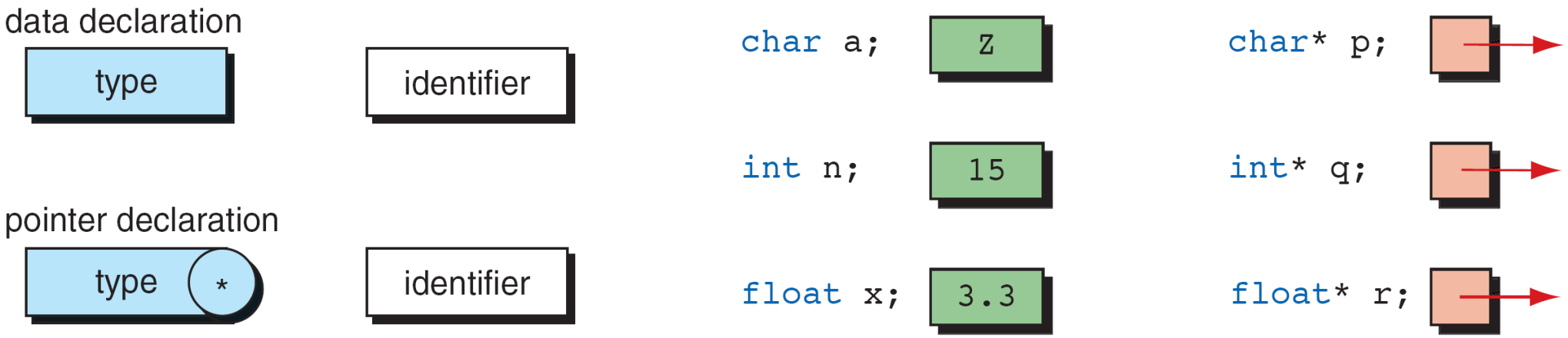
Pointer 선언, 사용

- Declaration

  int *ptr;

  → 정수형 데이터가 저장된 메모리의 위치(주소)를 저장할 수있는 포인터 변수

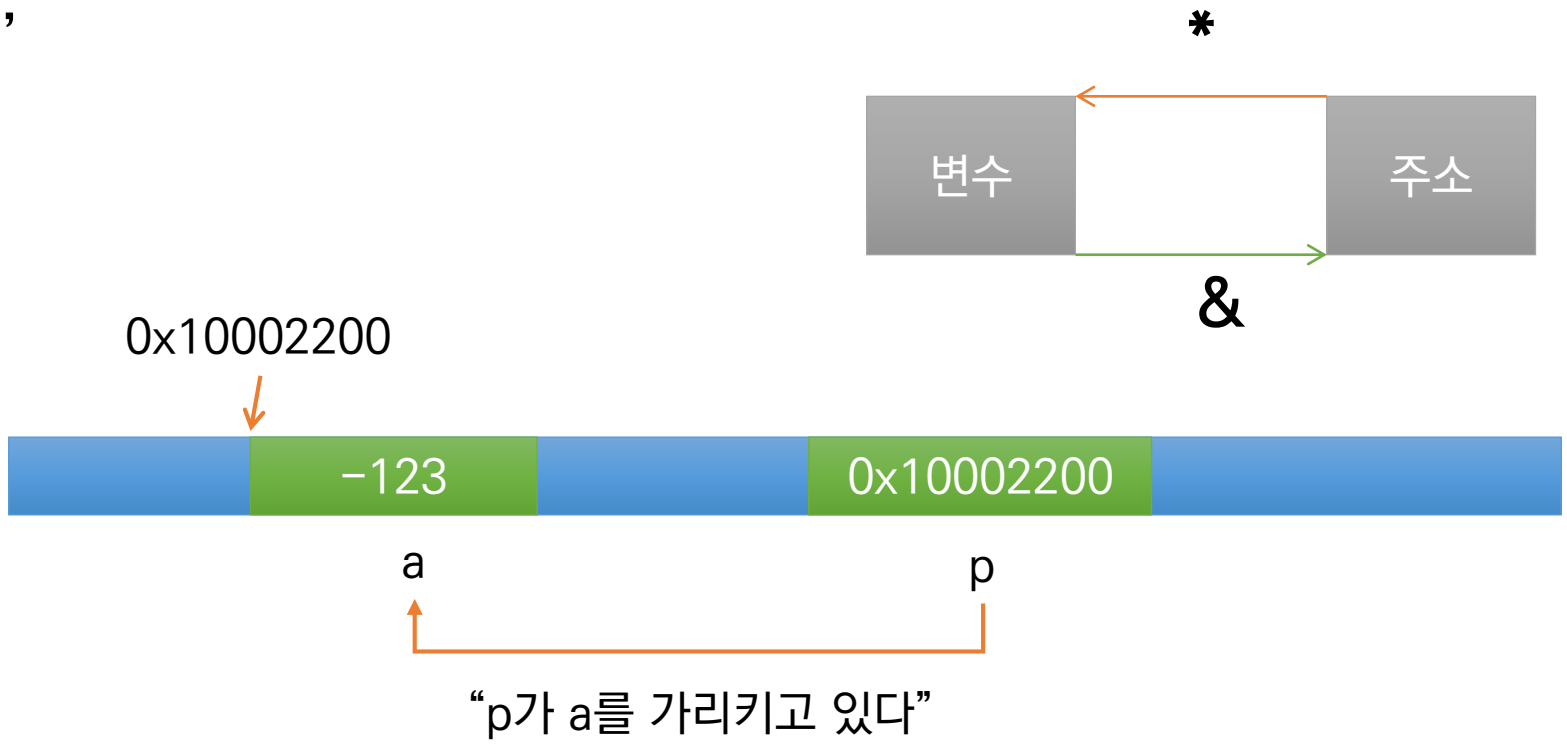
- 포인터의 타입 크기
  - 4 byte!(32 bit) ex) int : 4byte, char : 1 byte



Pointer 선언, 사용

int a = -123;

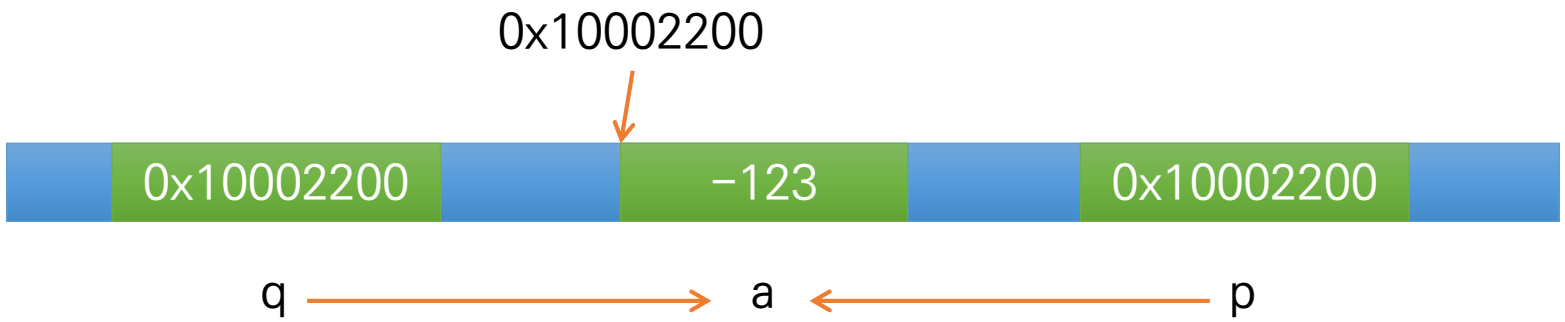
int *p = &a;



포인터 변수도 결국 ‘변수’이기 때문에 자신이 가리키는 주소를 메모리 어딘가에 저장하고 있다.

Pointer 선언, 사용

```
int a = -123;
int *p = &a;
int *q = p;  (or =&a;)
```



"p,q 둘 다 a를 가리키고 있다"
"a 변수의 데이터는 하나! a를 가리키는 포인터는 두 개!"

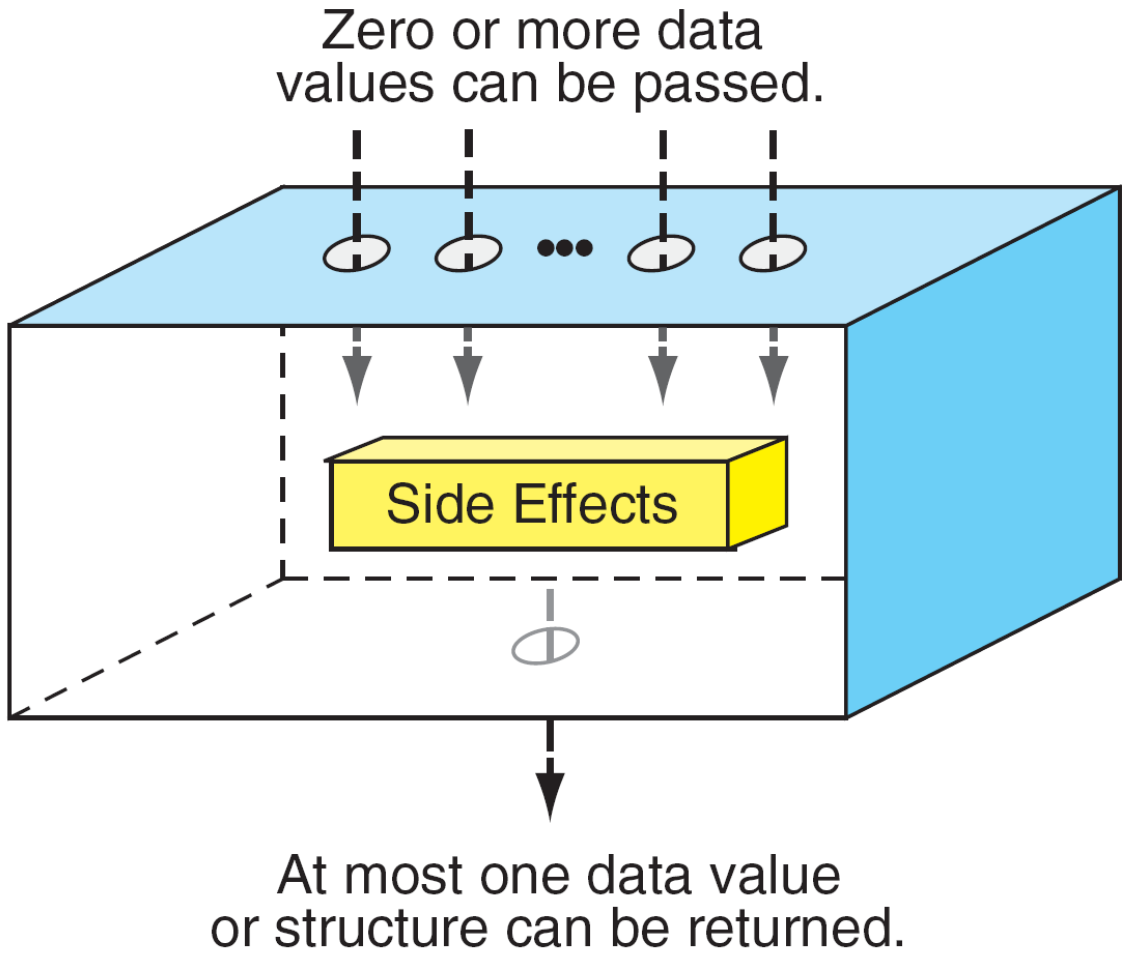
예제

```
int a = 14;
int* p = &a;

printf(“%p %d %d”, p, *p, a);
a = 20;
printf(“%p %d %d”, p, *p, a);
```

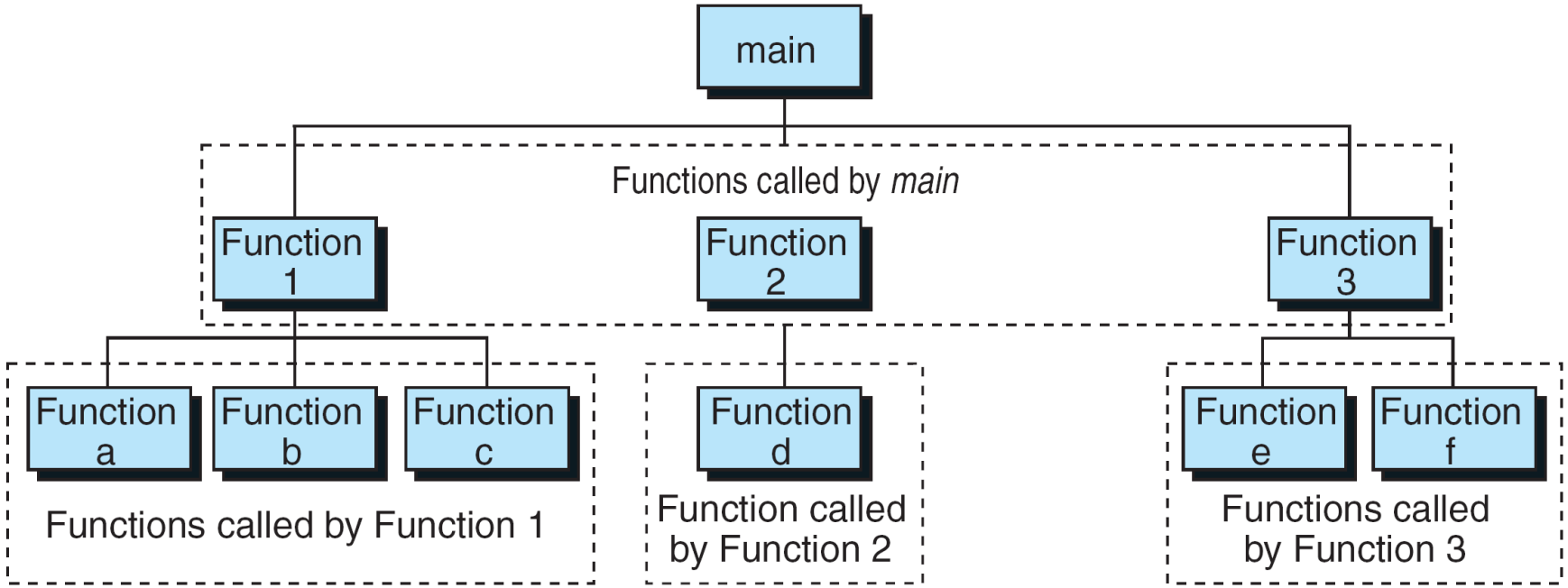
Results:
00135760 14 14
00135760 20 20

함수

함수

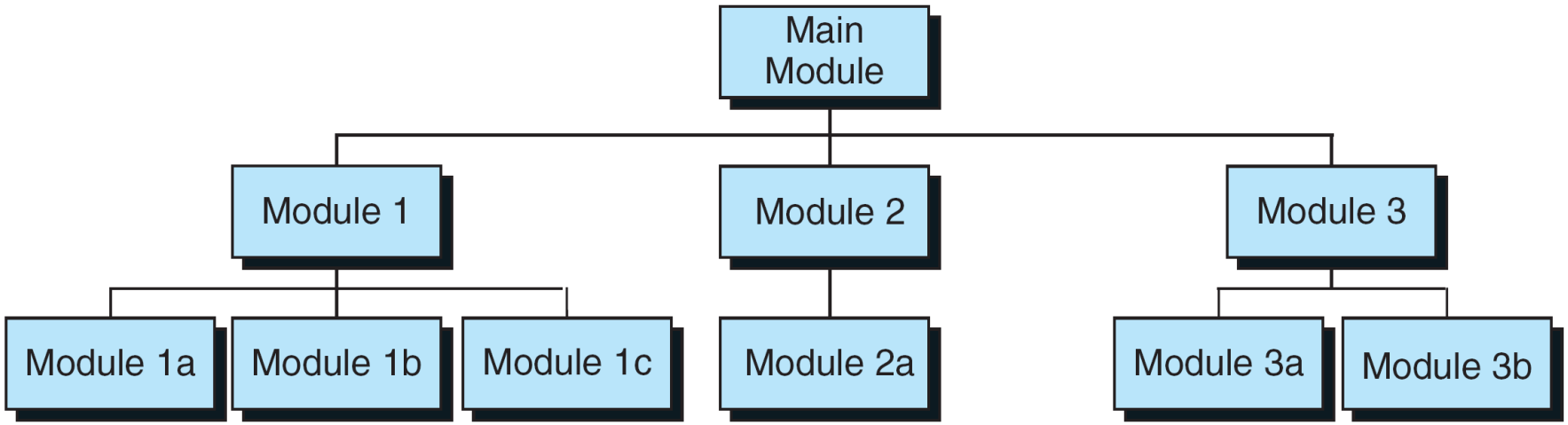

Zero or more data
values can be passed.
Side Effects
At most one data value
or structure can be returned.

C는 한 개 이상의 함수로 구성된다

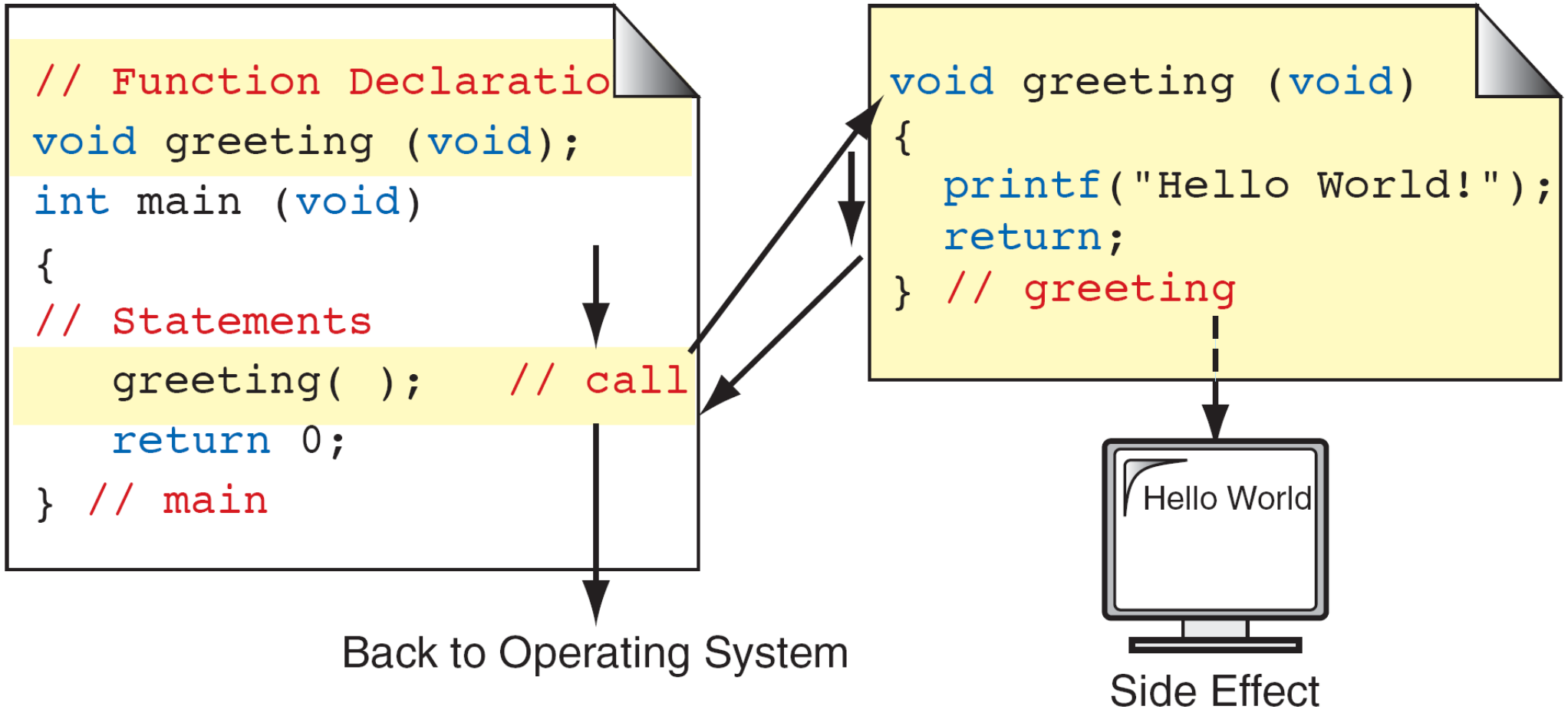


함수를 왜 만들죠

- 함수란 특정한 작업을 수행하도록 독립적으로 작성된 프로그램
- 프로그램에서 반복적으로 수행되는 부분을 함수로 작성하여 필요할 때마다 호출하여 사용
- 코드의 분할과 재사용



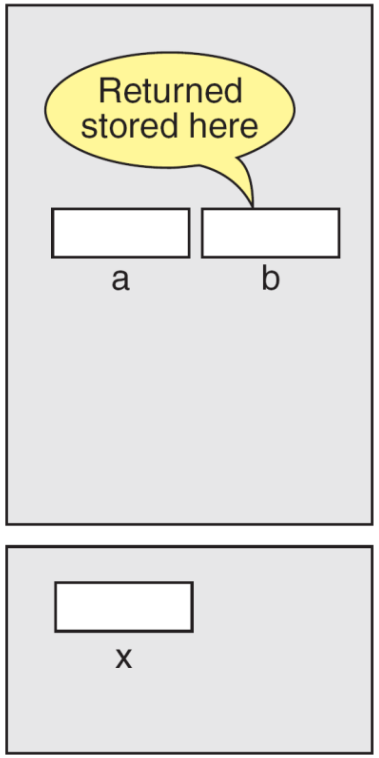
함수 선언, 정의, 사용


// Function Declaratio
void greeting (void);
int main (void)
{
// Statements
greeting( ); // call
return 0;
} // main
void greeting (void)
{
printf("Hello World!");
return;
} // greeting
Hello World
Back to Operating System
Side Effect

Calling a Function That Returns a Value

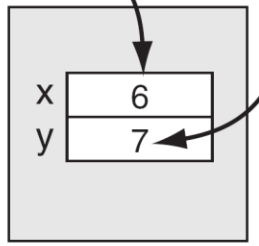
```c
// Function Declaration
int sqr (int x);
int main (void)
{
// Local Declarations
   int a;
   int b;
// Statements
   scanf("%d", &a);
   b  =  sqr (a);
   printf("%d squared: %d\n", a,  b);
   return 0;
}  // main
```

```c
int sqr (int x)
 {
// Statements
   return (x * x);
}  // sqr
```



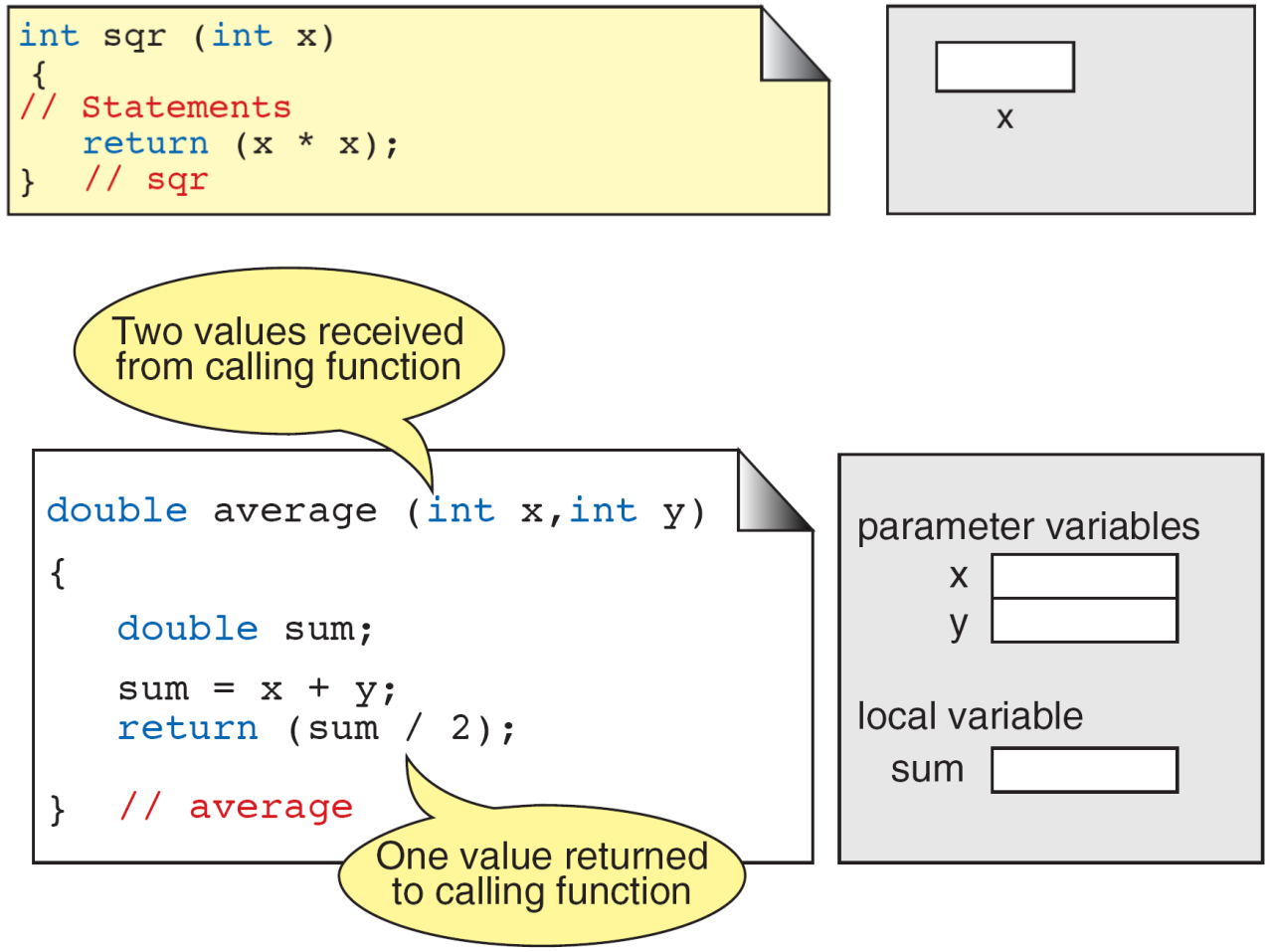

```c
// Function Declaration
int multiply (int multiplier, int multiplicand );

int main (void)
{
   int product;
   …
   product = multiply (6, 7);
   …
   return 0;
}  // main
```

```c
int multiply (int x, int y)
{
   return x * y;
}  // multiply
```

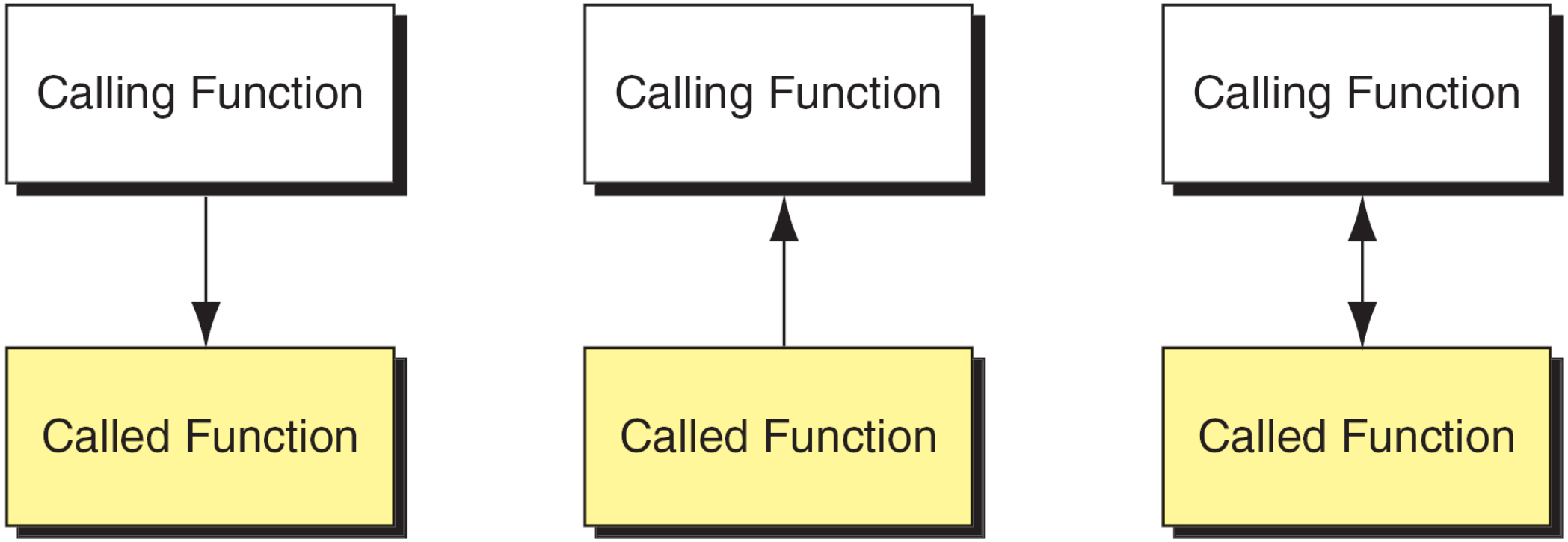


Function Definition

메모리 공간은 함수마다 따로따로



Inter-Function Communication

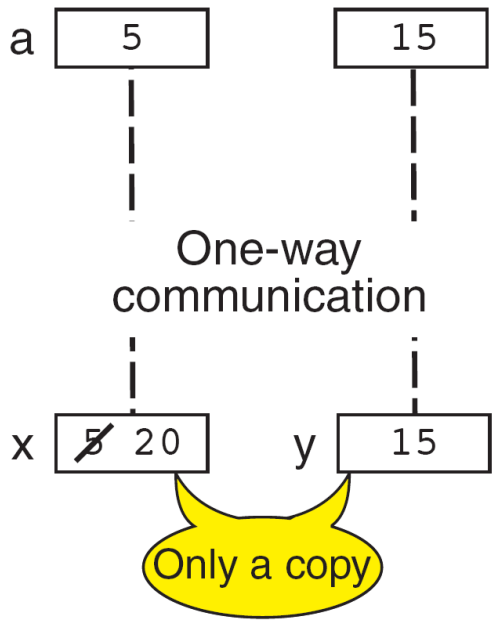


Call-by-Value

```
// Function Declaration
void downFun (int x, int y);
int main (void)
{
// Local Definitions
   int a = 5;
// Statements
   downFun (a, 15);
   printf("%d\n", a);
   return 0;
}  // main
```

prints 5

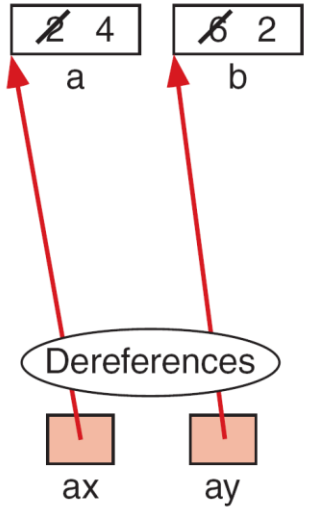
```
   void downFun (int x, int y)
   {
   // Statements
      x = x + y;
      return;
   }  // downFun
```



Call-by-Reference

```
// Function Declaration
void biFun (int* ax, int* ay);

int main (void)
{
// Local Definitions
   int  a = 2;
   int  b = 6;

// Statements
   …
   biFun (&a, &b);
   …
 return 0;
}  // main
```

```
void biFun (int* ax, int* ay)
{
   *ax = *ax + 2;
   *ay = *ay / *ax;
   return;
}  // biFun
```



SWAP

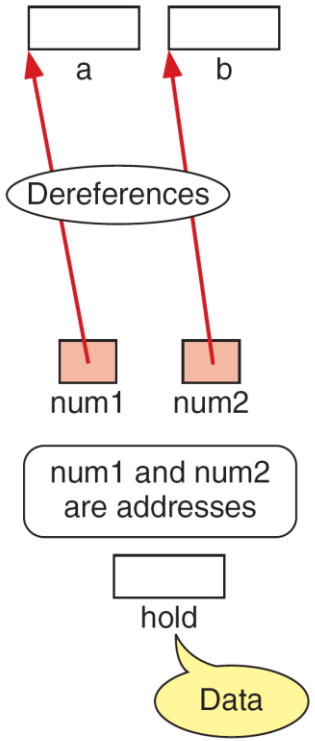
```
// Function Declarations
void exchange (int* num1, int* num2);

int main (void)
{
// Local Definitions
   int  a;
   int  b;

// Statements
   …
   exchange (&a, &b);
   …
 return 0;
}  // main
```

Note that the type includes an asterisk.

Address operators

```
void exchange (int* num1, int* num2)
{
// Local Definitions
   int  hold;

// Statements
   hold   =  *num1;
   *num1  =  *num2;
   *num2  =   hold;
   return;
}  // exchange
```

Note the indirection operator is used for dereferencing.



Scope

```
/* This is a sample to demonstrate scope. The techniques
   used in this program should never be used in practice.
*/
#include  <stdio.h>
int fun (int a, int b);                          Global area

 int main (void)
 {
   int   a;                                      main's area
   int   b;
   float y;
   …
      {  // Beginning of nested block
        float a = y / 2;
        float y;                                 Nested block
        float z;                                 area
        …
        z = a * b;
        …
      }  // End of nested block
   …
 }  // End of main

int fun (int i, int j)
{
   int a;                                        fun's area
   int y;
   …
}  // fun
```

- 중괄호로 싸여져 있는 코드 블럭, 코드 범위
- 한 Scope 내에서 선언된 변수는 그 Scope 범위 안에서만 존재한다.
- 자기보다 상위 Scope의 변수는 볼 수 있지만 하위 Scope의 변수는 볼 수 없다.
- Scope별로 변수를 만들 수 있다.
- 바깥 Scope, 지금 Scope 에 같은 이름의 변수가 있으면 현재 Scope 에 있는 변수를 본다.

마치기 전에

Tip

- 수업시간 자료 – 초록색 박스에 있는 키워드와 설명
- 책에 나오는 예시 코드 꼭 보기!
- 시험에서 코드를 주고 틀린 부분을 고치거나, 함수를 비워놓고 손코딩 하는 문제들이 나온다
- 중간중간 나오는 실수하기 쉬운 예시들 기억해두기

- **이런것도 될까? 싶을 때는 해보는게 최고!**
- **책에 있는 예제 코드를 한번씩 직접 코딩해 보자.**

다음 시간

- 조건문
- 반복문