전산 SMP 3주차

2015. 10. 06

CSE 12 김범수

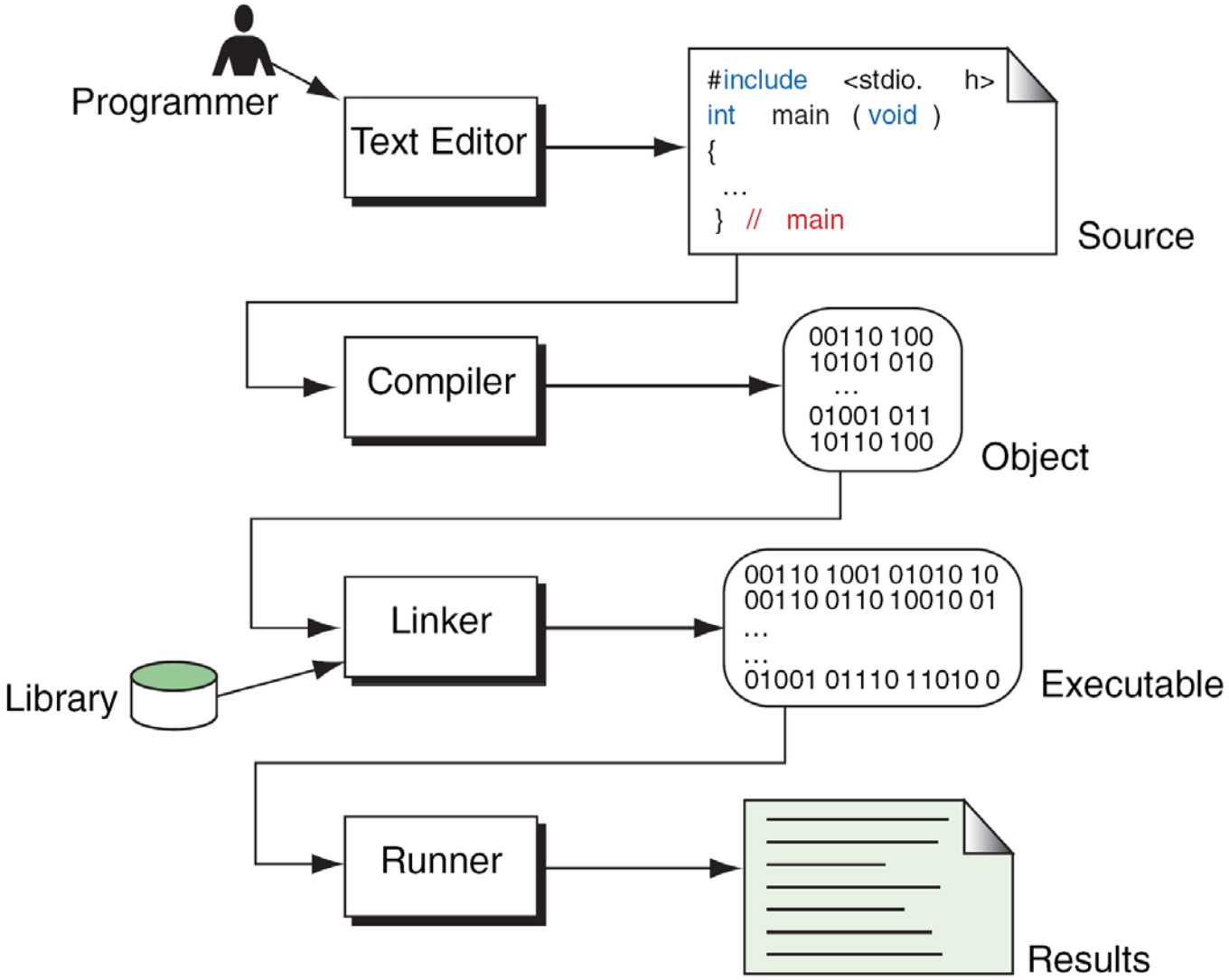
bskim45@gmail.com

Special thanks to 박기석 (kisuk0521@gmail.com)

지난시간 복습

(추석 == 포맷)?

C 언어에서 프로그램(Executable)으로



기본 자료형(Primitives)의 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

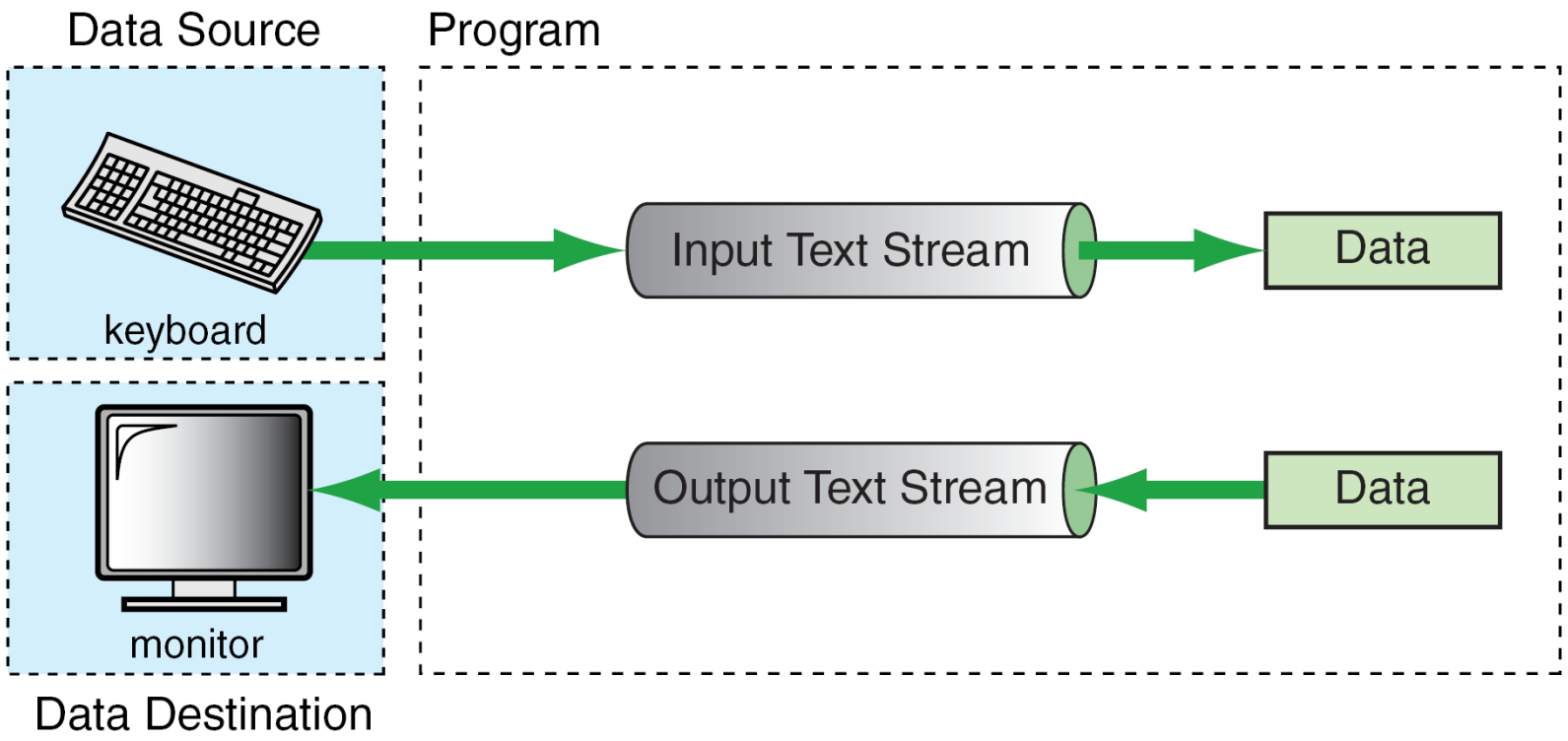
형 변환 (Type Conversion)

- Implicit Conversion
  - `int a = 3.14;  // a = 3`

- Explicit Conversion (casting)
  - 프로그래머가 인위적으로 타입을 바꿈
  - `double a = (double) 4/3  // a = ??`
  - `double b = (double) 4 / (double) 3  // b = ??`

Standard I/O Stream

- Header **`<stdio.h>`**: Standard input/output libraries



printf

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

printf(“%d  %d  %d\n”, a, b, c);
```

서식문자(Format String) : 데이터의 출력 형태를 지정

ex. %d : 부호 있는 10진 정수로 출력하라!

scanf

```
int a;
char b;

scanf(“%d %c”, &a, &b);
```

scanf로 값을 받기 위해서는 받는 위치, 즉 주소값을 인자로 주어야 한다.
이유는 다음 시간에. 일단 외우자.

& 가 변수 이름 앞에 붙을 시 그 변수가 가리키는 주소를 의미한다.

A : 변수 이름 / &a : 변수 a 의 주소

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

연산자의 종류

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

오늘 할 것

- 포인터 맛보기
- 함수

시간이 된다면…

- 조건문
- 반복문

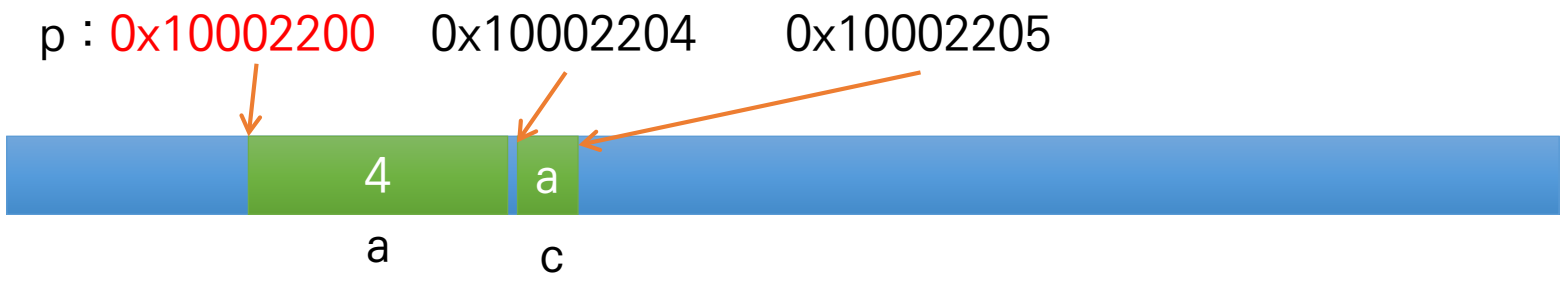
Pointer 맛보기

포인터의 개념

Pointer란?

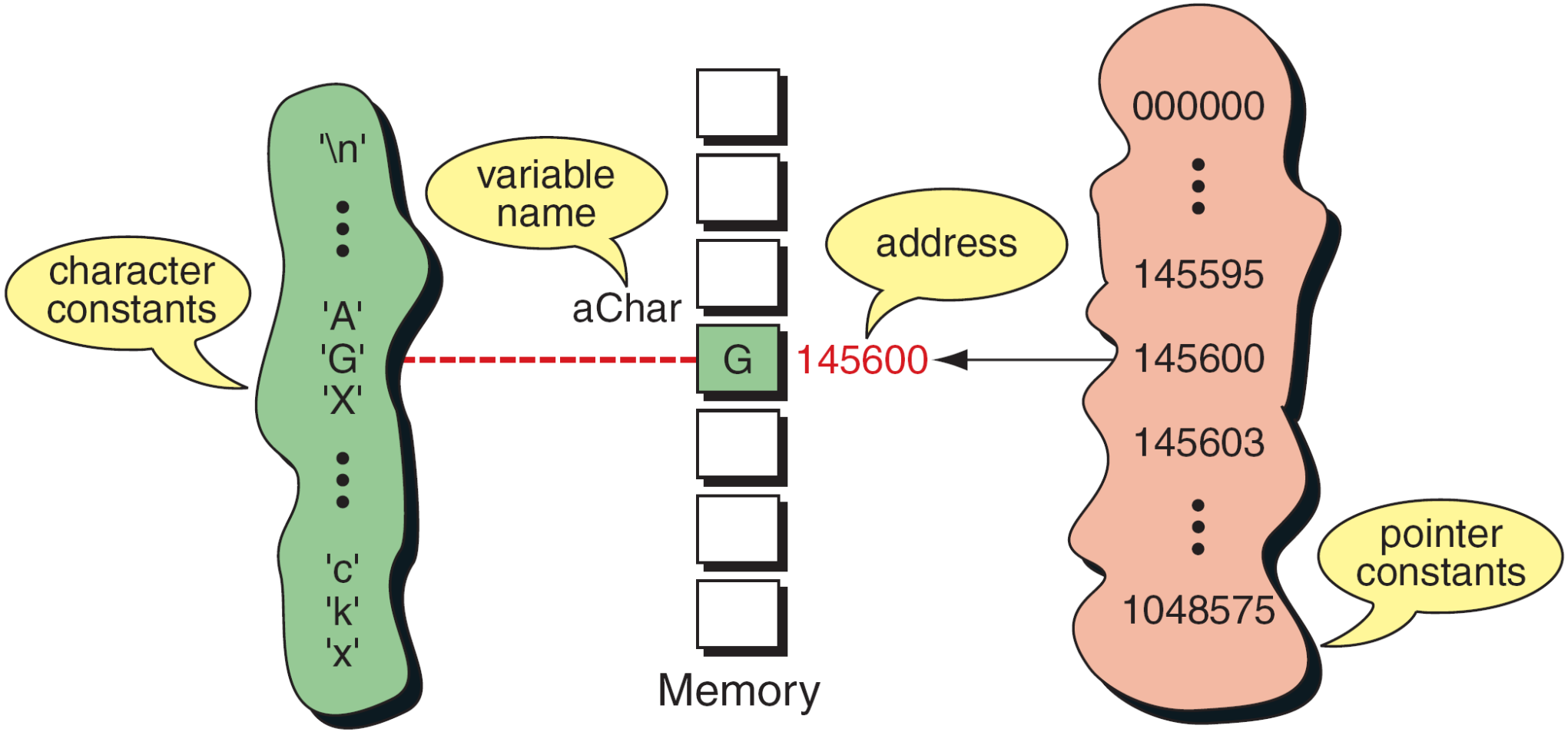
- 데이터 접근에 사용되는 주소를 저장하는 타입 - 포인터도 타입이다

- 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

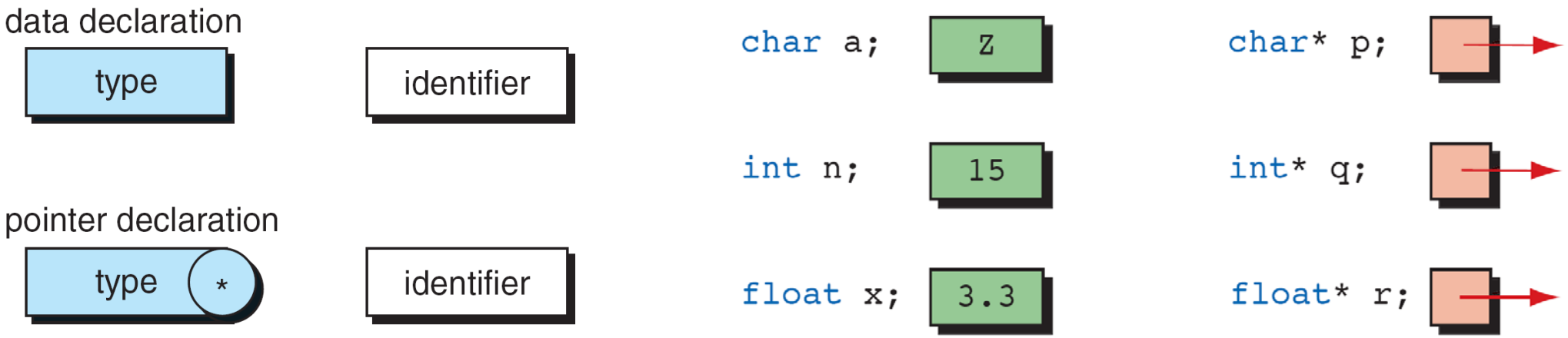
Pointer 선언, 사용

- 선언(Declaration)

  int *ptr;

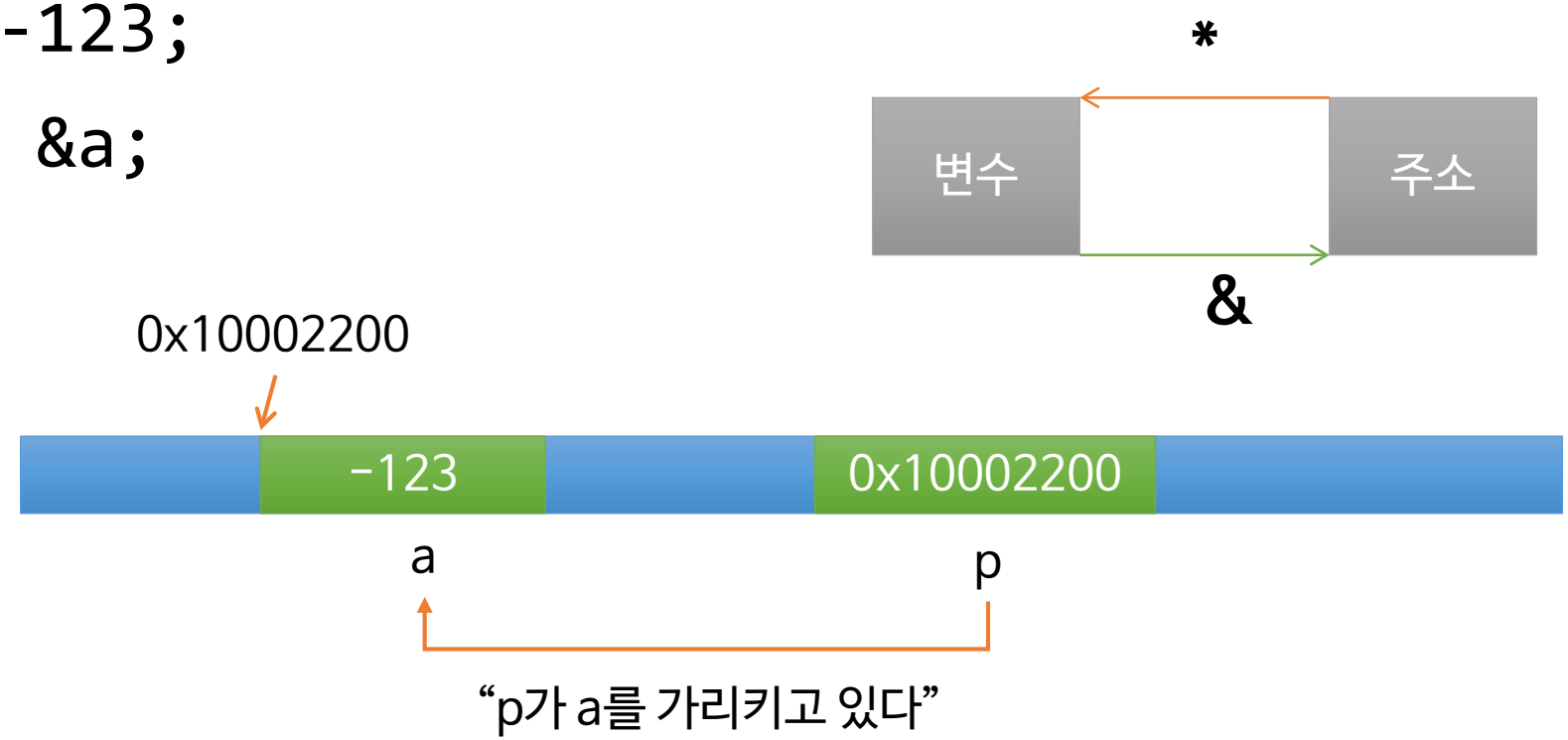
  → 정수형 데이터가 저장된 메모리의 위치(주소)를 저장할 수있는 포인터 변수

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



Pointer 선언, 사용

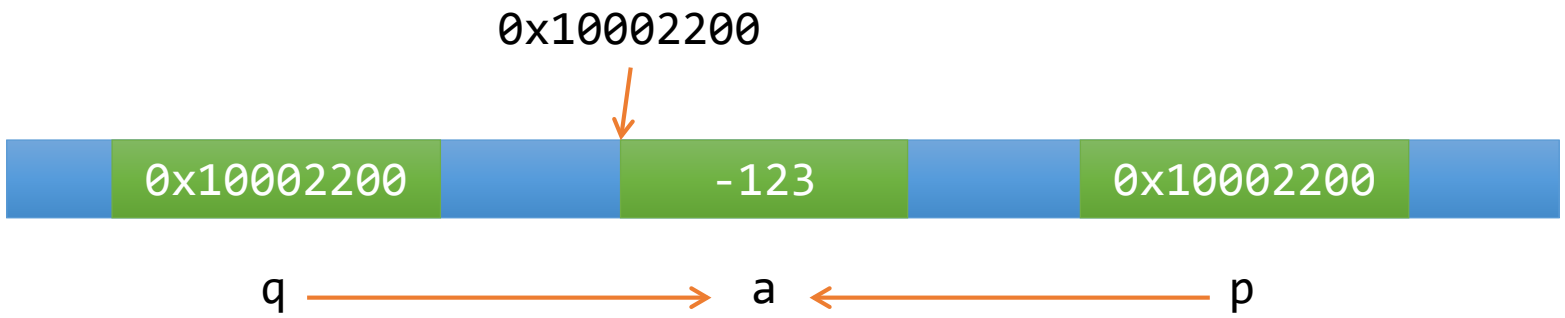
```
int a = -123;
int *p = &a;
```



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;)
```



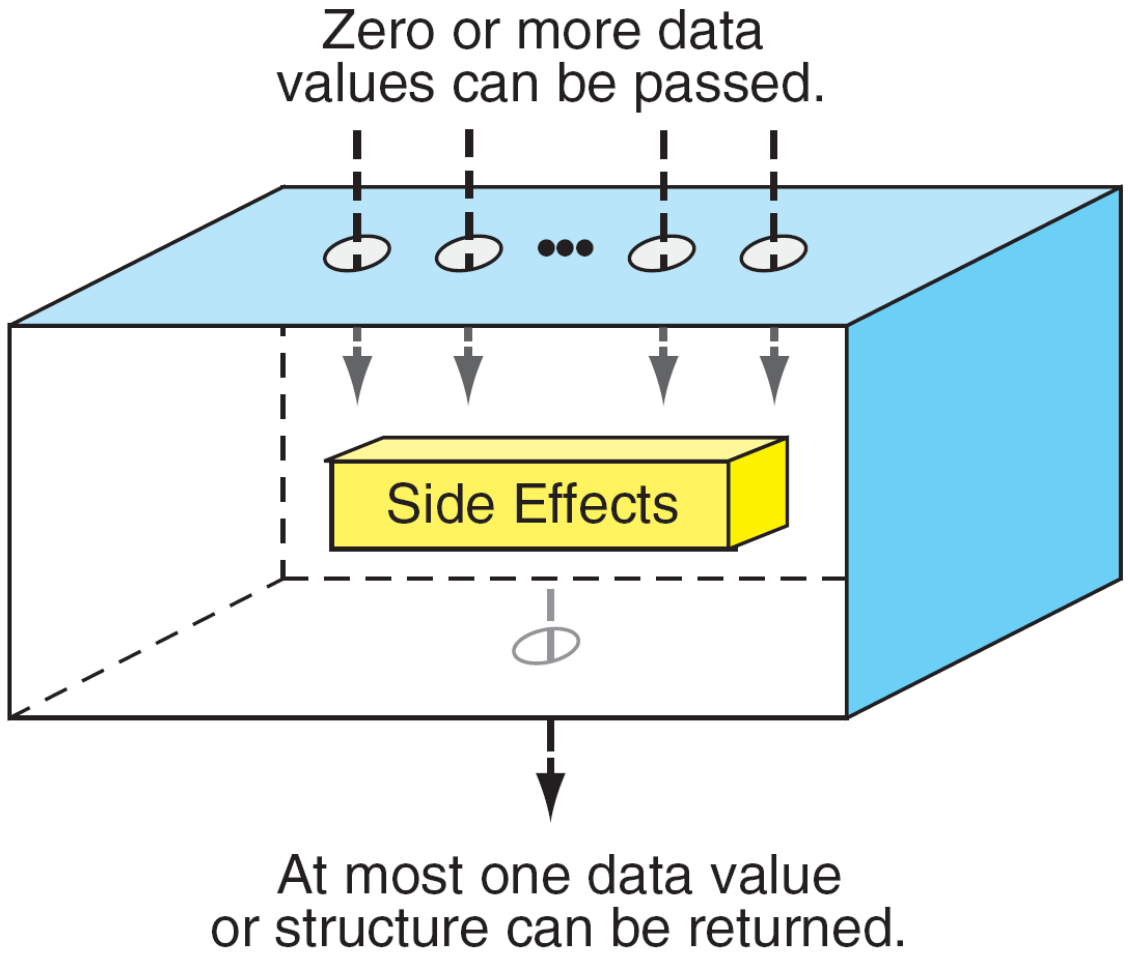
"p, q 둘 다 a를 가리키고 있다"
"a 변수의 데이터는 하나! a를 가리키는 포인터는 두 개!"

예제

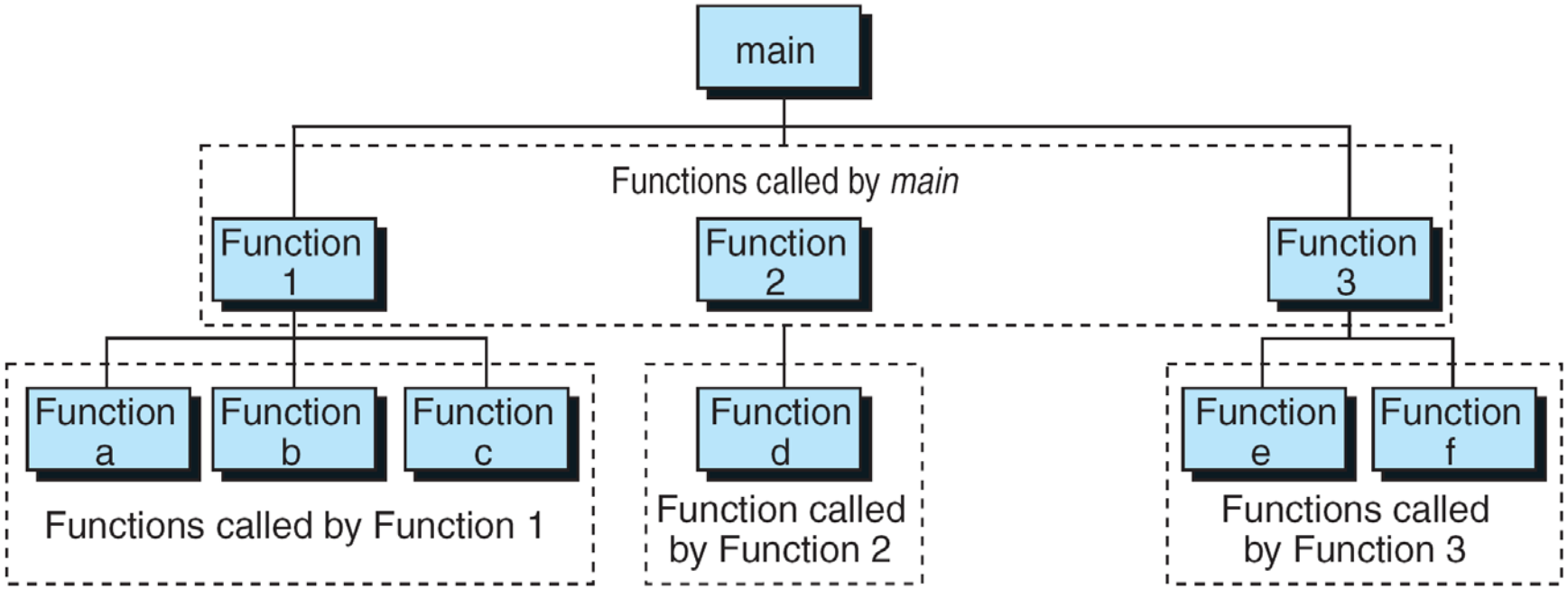
```
int a = 14;
int* p = &a;

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

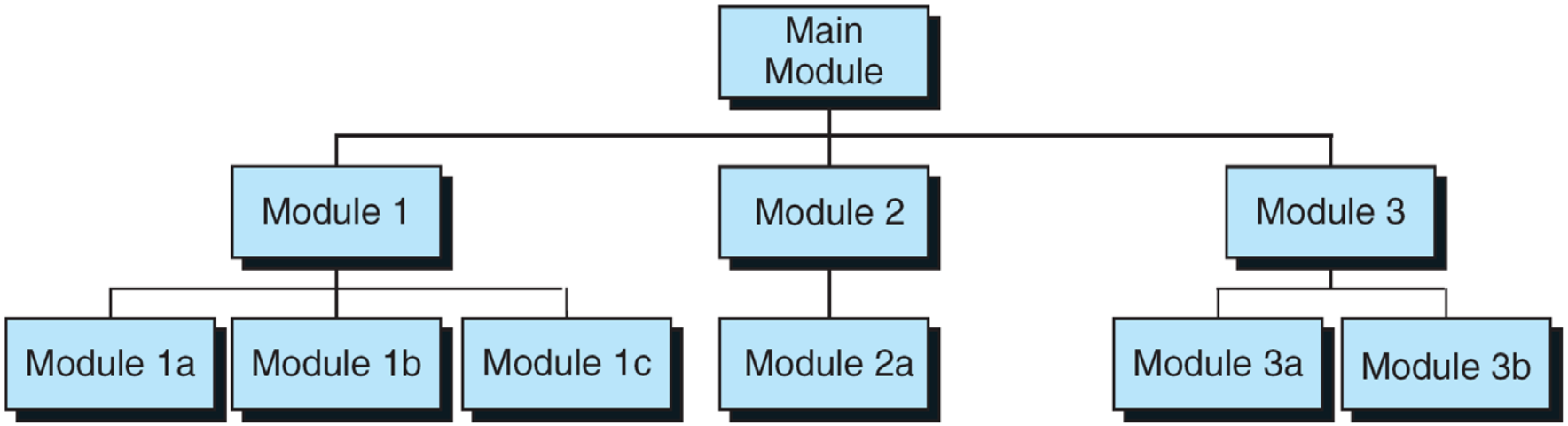

Zero or more data
values can be passed.
Side Effects
At most one data value
or structure can be returned.

C는 한 개 이상의 함수로 구성된다

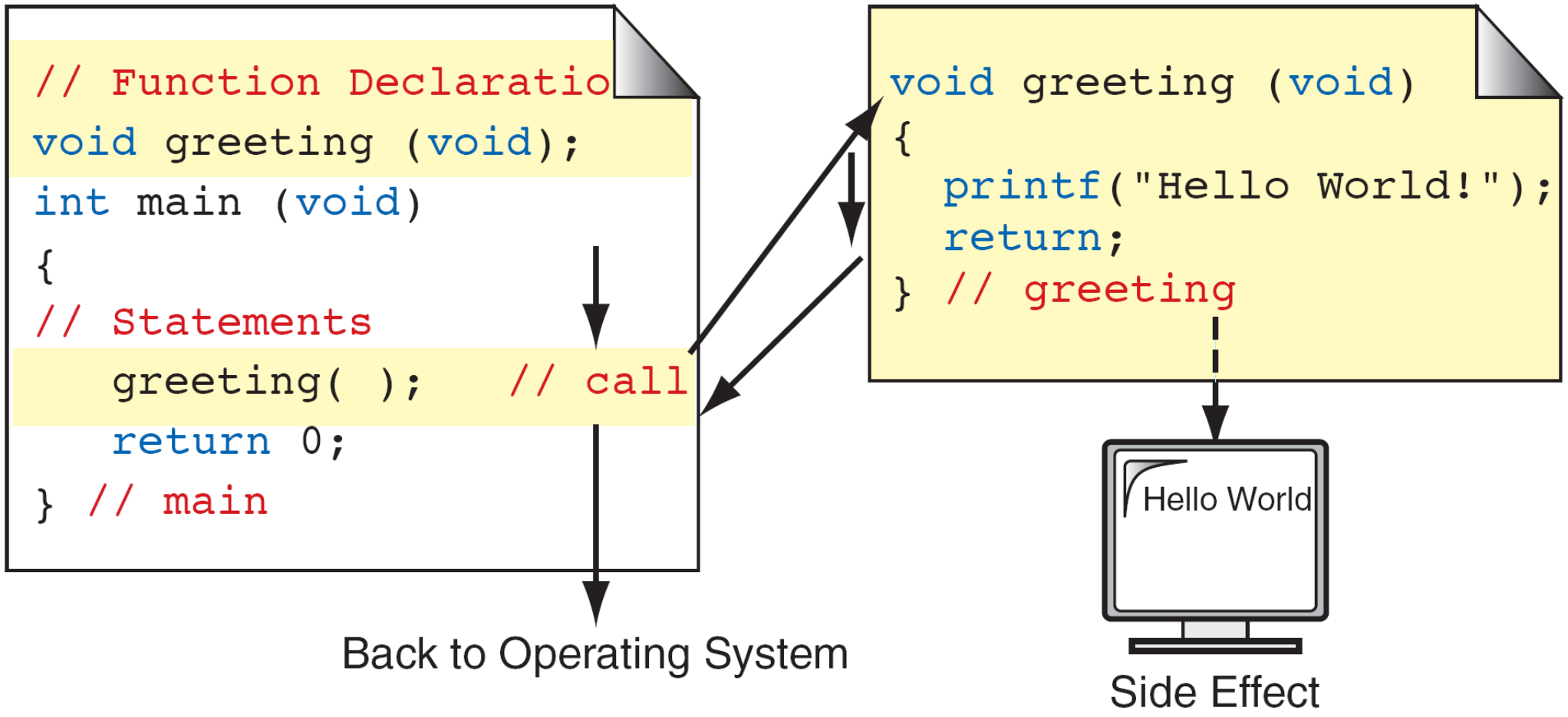


함수를 왜 만들죠

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



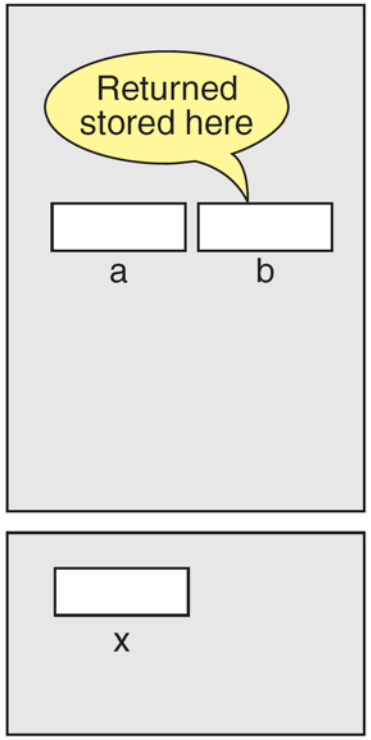
함수 선언, 정의, 사용


// 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

```
// 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
```

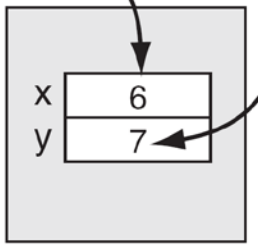
```
int sqr (int x)
 {
// Statements
   return (x * x);
}  // sqr
```




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

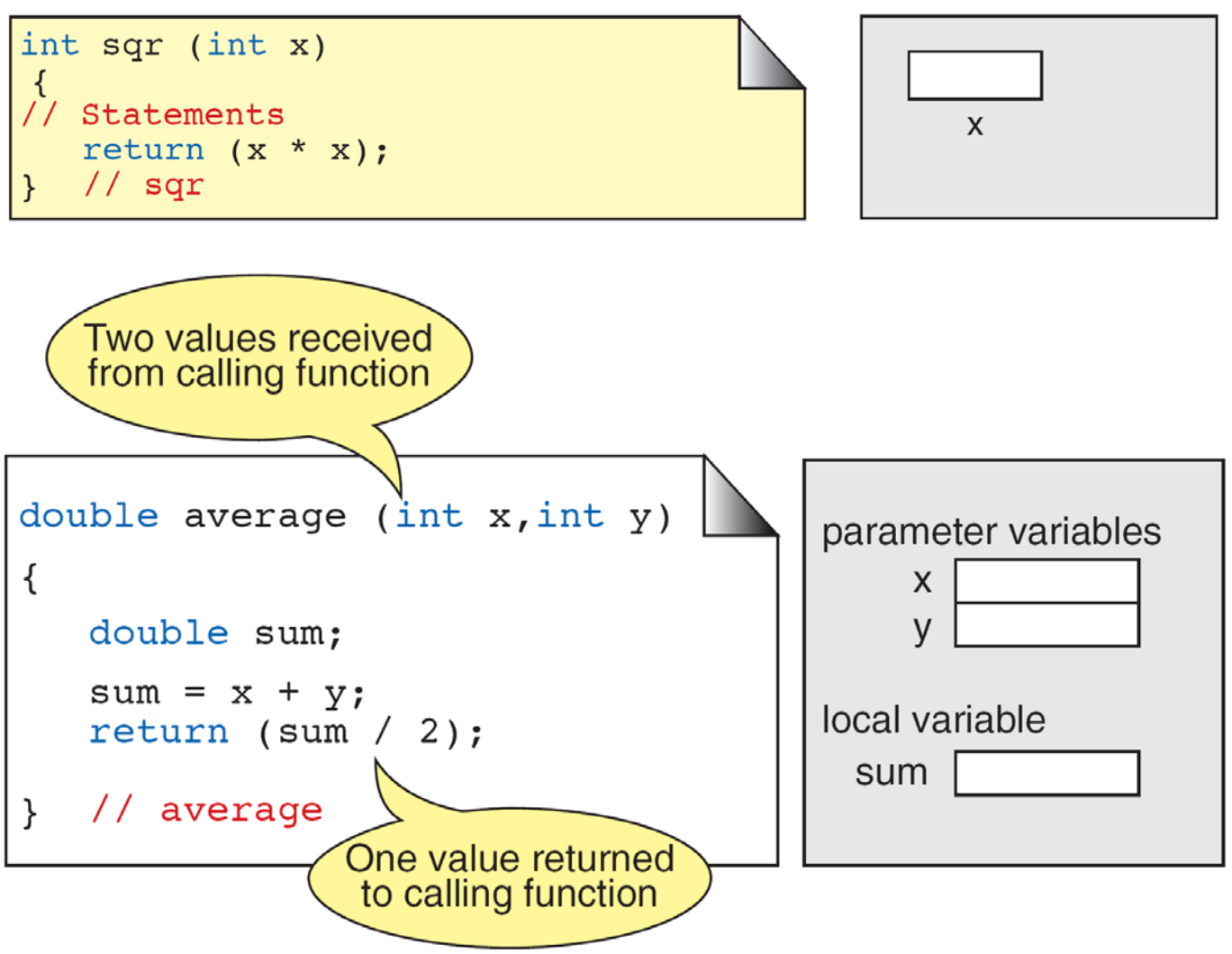
42

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

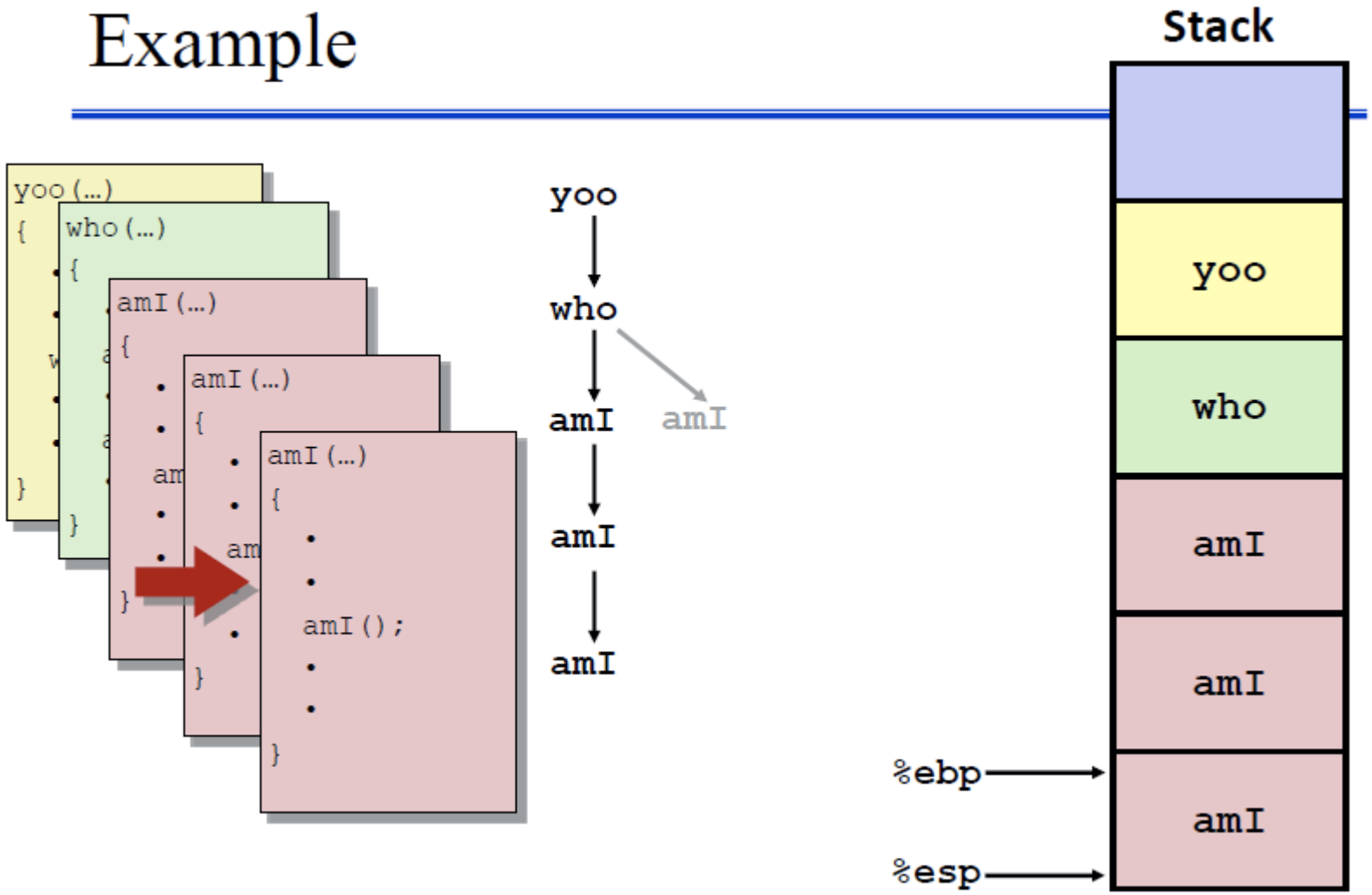


Function Definition

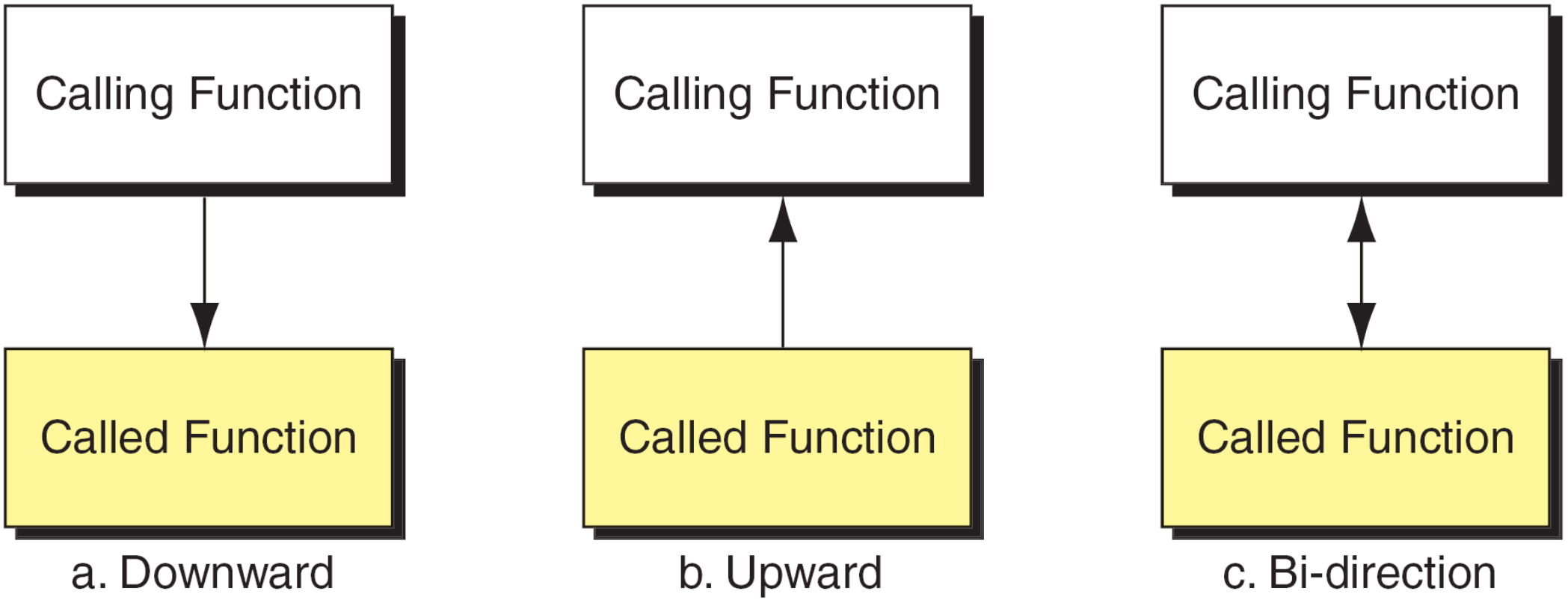
메모리 공간은 함수마다 따로따로



순서대로 메모리에


Example
yoo (...)
{
who (...)
{
amI (...)
{
amI (...)
{
amI (...)
{
amI();
}
yoo
who
amI
amI
amI
amI
Stack
yoo
who
amI
amI
amI
%ebp
%esp

Inter-Function Communication

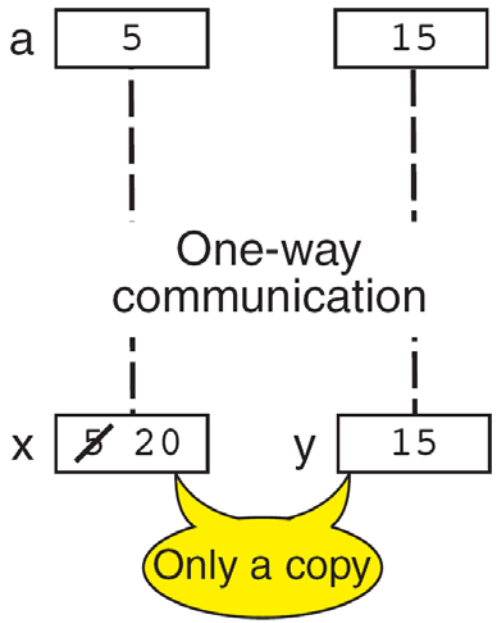


Call-by-Value

```
// 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

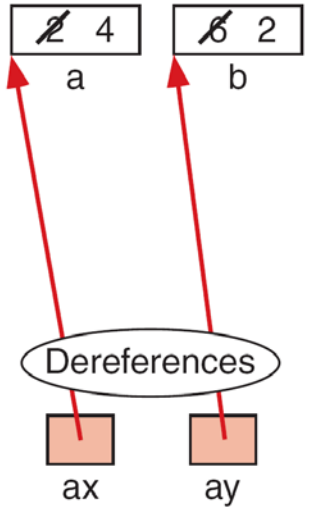
```
   void downFun (int x, int y)
   {
   // Statements
      x = x + y;
      return;
   }  // downFun
```



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

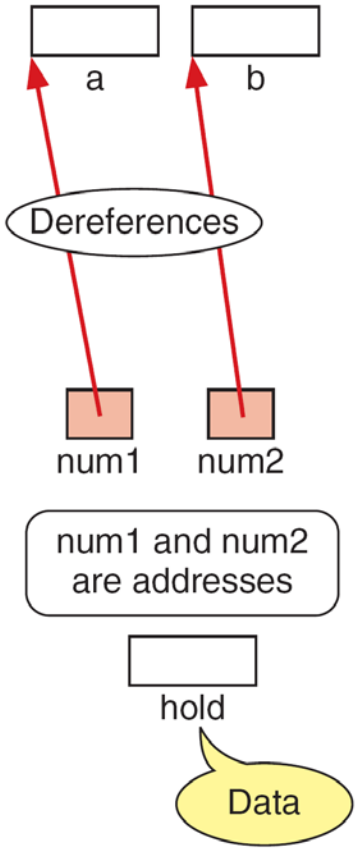
```
// Function Declarations
void exchange (int* num1, int* num2);

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

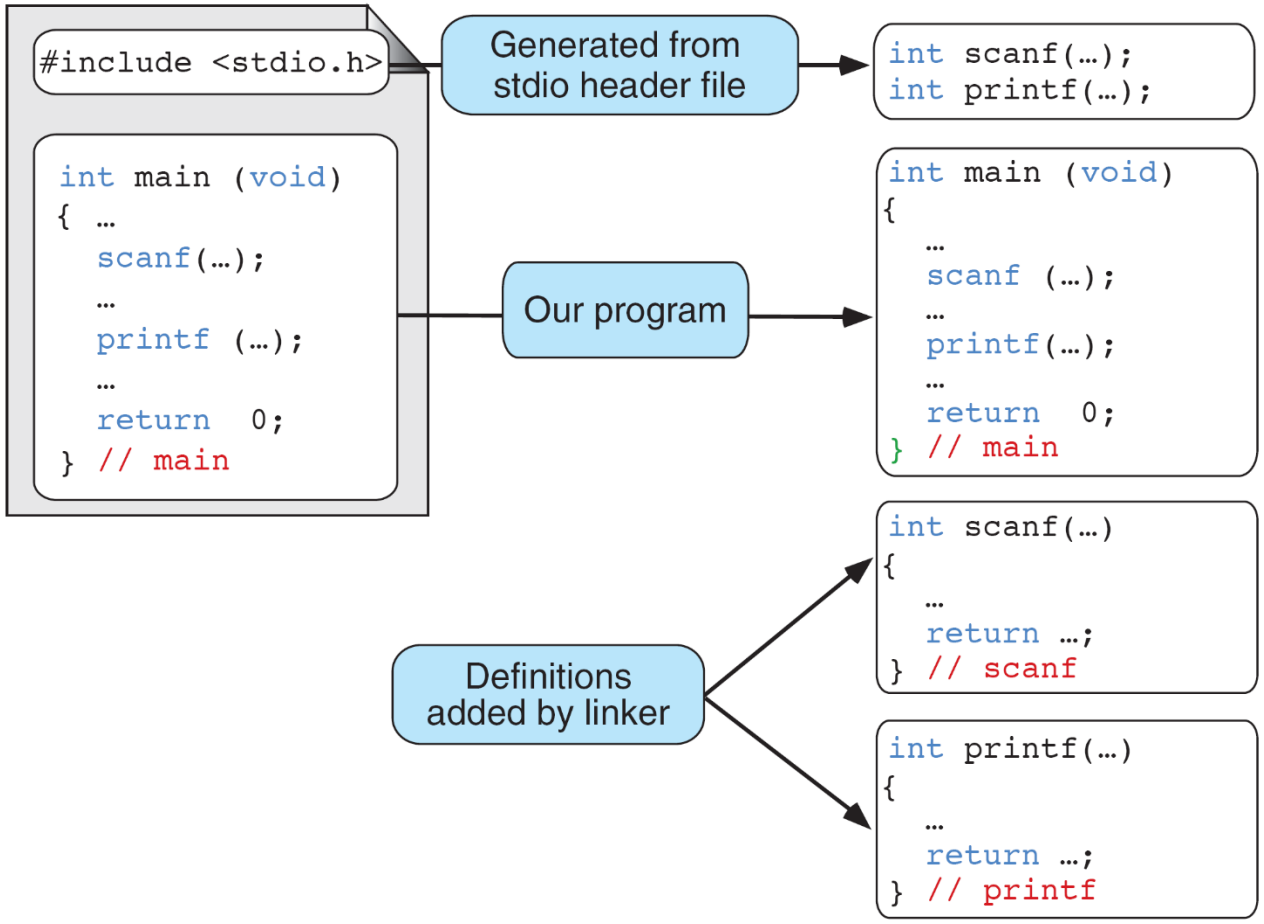
```
void exchange (int* num1, int* num2)
{
// Local Definitions
   int  hold;

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

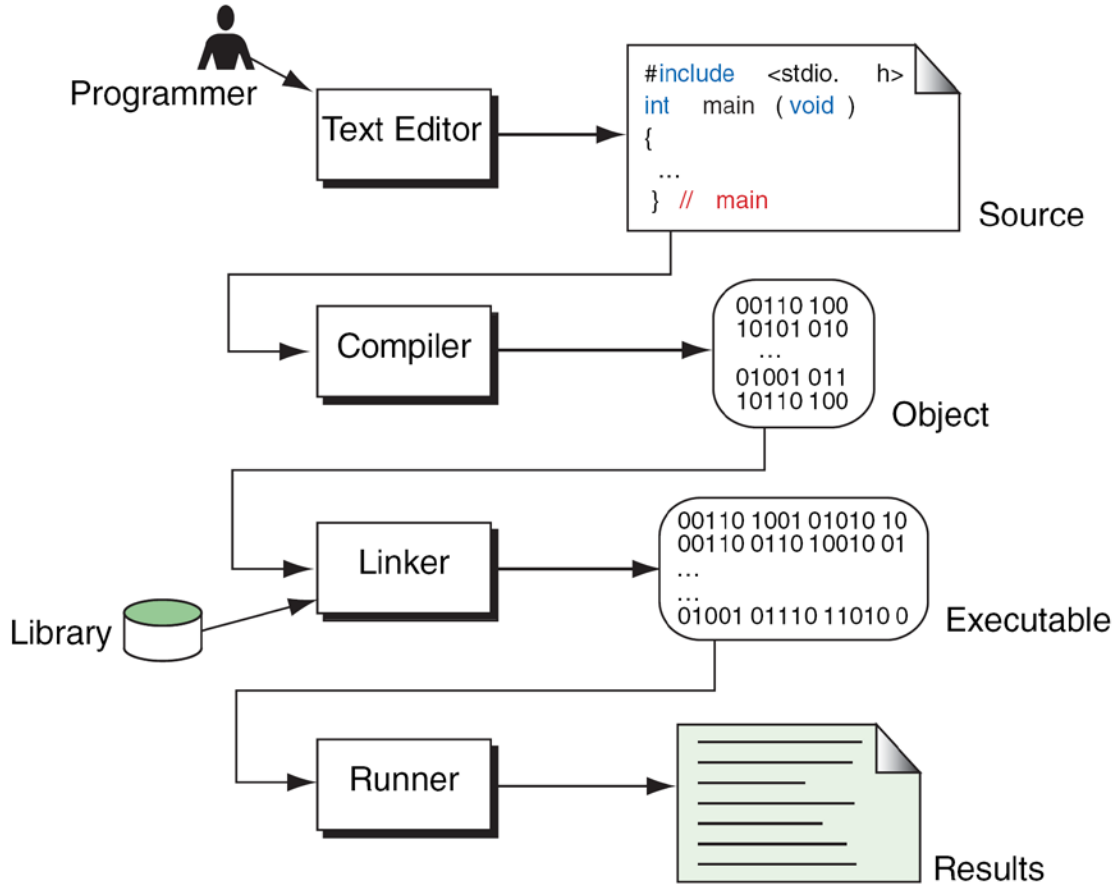
Note the indirection operator is used for dereferencing.



Standard Functions

Library Functions and the Linker





Math Functions `<math.h>`

- Absolute value functions: abs
- Floor, Ceiling functions: floor, ceil
- Round functions: round
- Power function: pow
- Square root function: sqrt

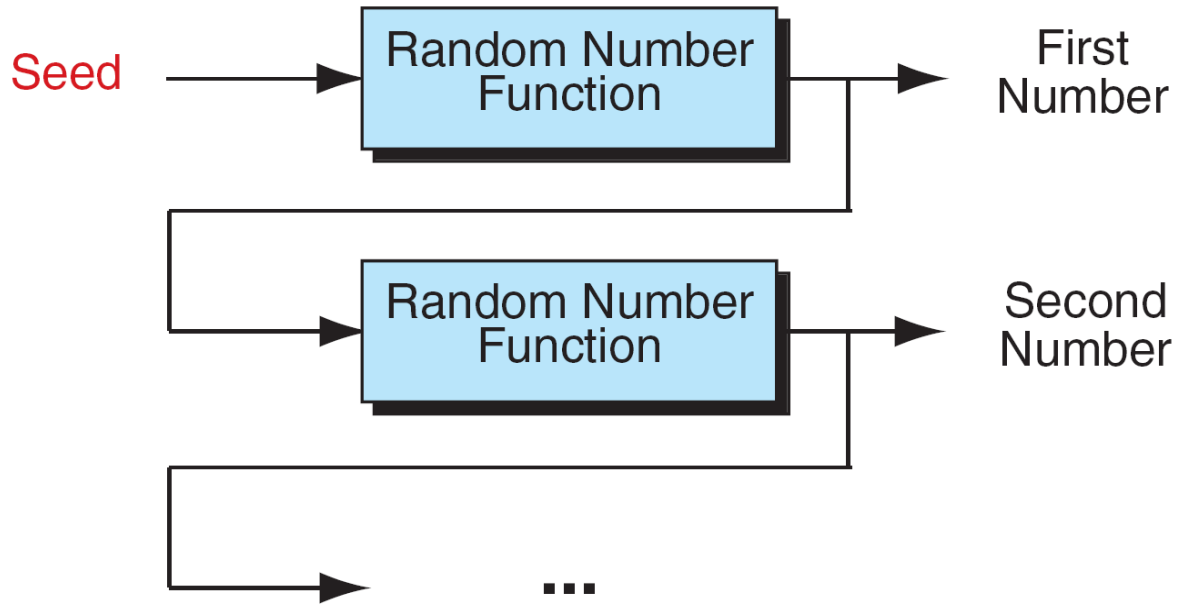
- http://www.cplusplus.com/reference/cmath/

Generating Random Numbers <stdlib.h>

- C Standard General Utilities Library
- srand() 를 통해 시드(Seed) 생성

- `srand (997);`
- `srand (time(NULL));`

- `int rand (void);`



Creating Temporal Random Numbers

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

int main(void) {
        srand(time(NULL));

        printf("%d\n", rand());
        printf("%d\n", rand());
        printf("%d\n", rand());

        return 0;
}
```

```
Results:
First Run
    9641
    16041
    6350
Second Run
    31390
    31457
    21438
```

Creating Pseudorandom Numbers

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

int main(void) {
        srand(997);

        printf("%d\n", rand());
        printf("%d\n", rand());
        printf("%d\n", rand());

        return 0;
}
```

```
Results:
First Run
    10575
    22303
    4276
Second Run
    10575
    22303
    4276
```

Generating Random Numbers in (10…20)

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

int main(void) {
	int range = (20 - 10) + 1;
	srand(time(NULL));

	printf("%d\n", rand() % range + 10);
	printf("%d\n", rand() % range + 10);
	printf("%d\n", rand() % range + 10);

	return 0;
}
```

```
Results:
10 11 16
```

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
```

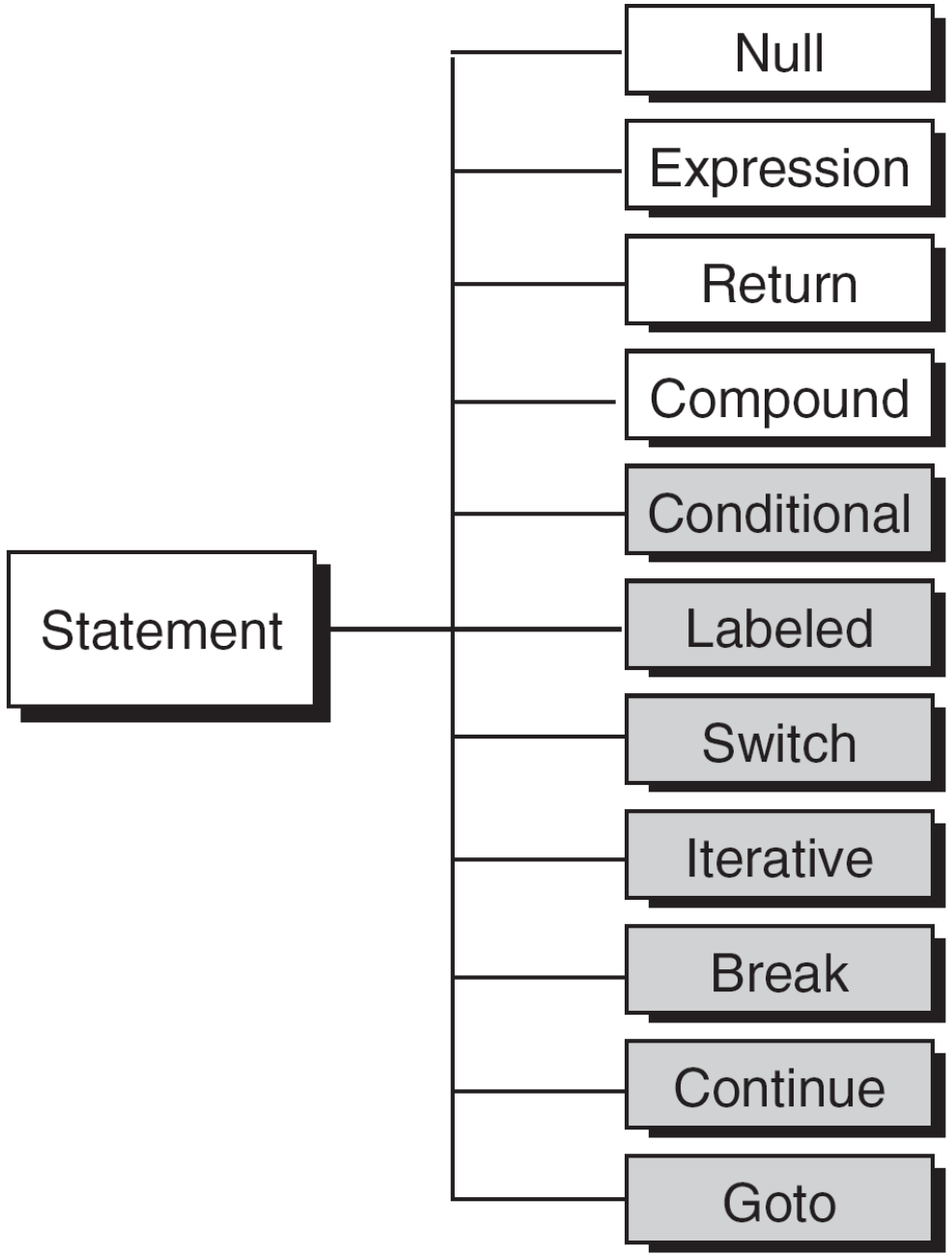
- 중괄호로 싸여져 있는 코드 블럭, 코드 범위
- Scope별로 변수를 만들 수 있다.
- Scope 내에서 선언된 변수는 해당 Scope 안에서만 존재한다.
- 자기보다 상위 Scope의 변수는 볼 수 있지만 하위 Scope의 변수는 볼 수 없다.
- 바깥 Scope, 지금 Scope 에 같은 이름의 변수가 있으면 현재 Scope 에 있는 변수를 본다.

Statement

명령문

Statement

- Null statement
- Expression statement
- Return statement
- Compound statement



Null Statement

- Just a semicolon
- Example

```
;                       // null statement
```

- They do nothing, but are still valid syntactical objects.

Expression Statement

- A statement consisting of an expression
- Example
  expression;

  a = 2;

  b = c = 3;

Return Statement

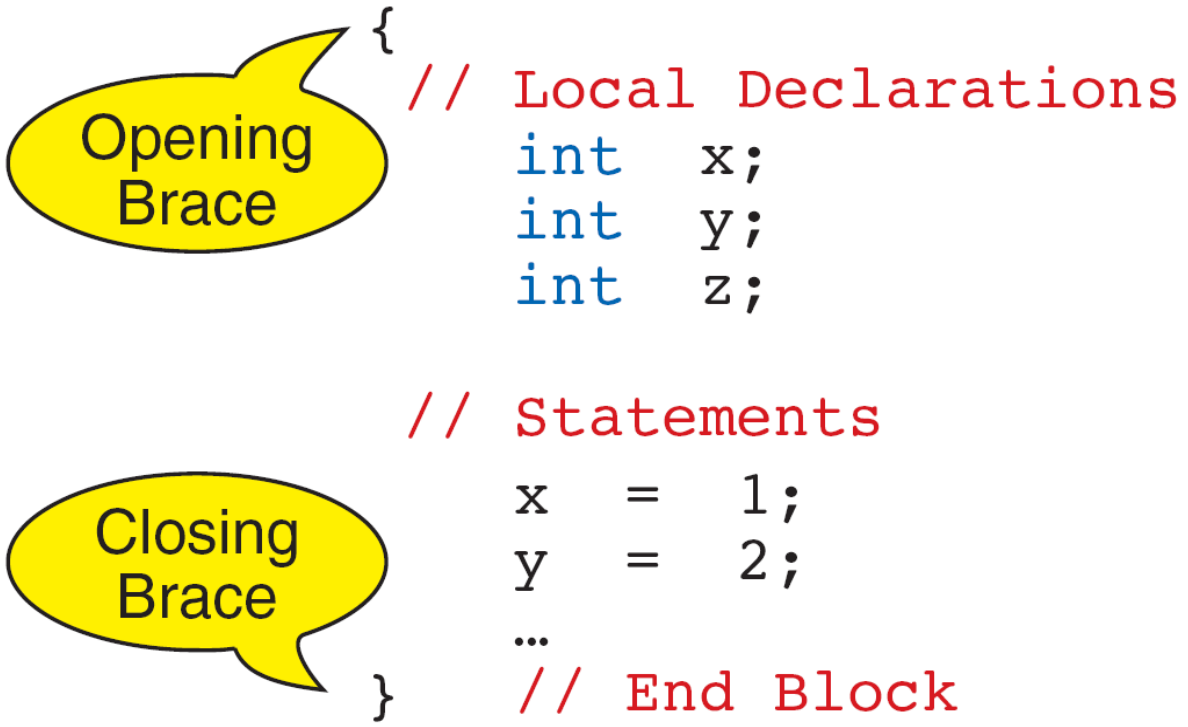
- A return statement terminates a function.

- Example

  return;

  return expression;

Brace ;



The compound statement does not need a semicolon.

The Role of the Semicolon

- Every declaration in C is terminated by a semicolon
- Most statements in C are terminated by a semicolon

Statements and Defined Constants

```
#define TAX_RATE 0.825

tax = TAX_RATE * amount;
```

조건문

조건문 (Selection Statement)

- 조건 (Condition)에 따라서 선택적으로 프로그램을 진행
- 프로그램의 Flow를 조종

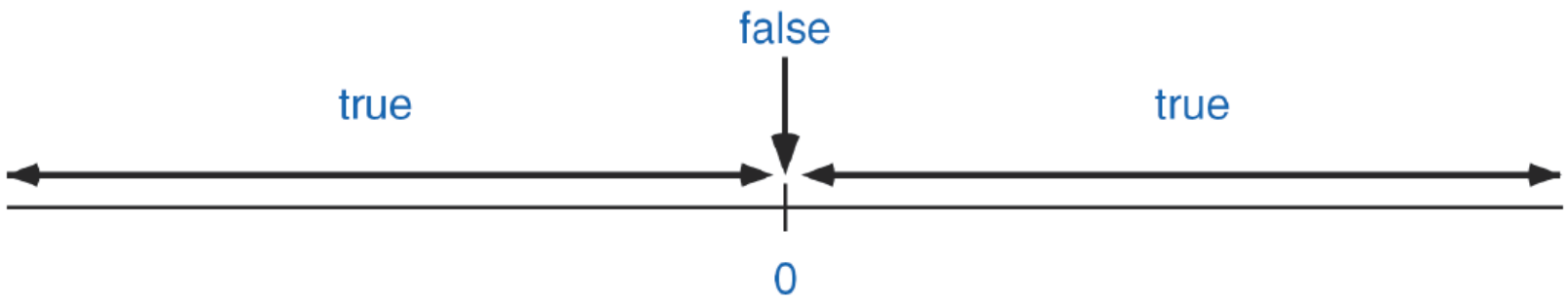
2-Way Selection
  - 두 가지 선택지 중에 한 가지

Multi-Way Selection
  - 여러 가지 선택지 중에 한 가지(혹은 그 이상)

- 조건문 안에 얼마든지 또 조건문을 넣을 수 있다. (nested)

조건 Condition (TRUE / FALSE)

- 조건문과 반복문 등에서 '조건 검사'를 할 때 사용
- C에서는 0을 FALSE, 0이 아니면 TRUE
- ex. if(0.4)는 TRUE로 취급



복잡한 조건들 – 논리 연산자

- ! - NOT, && - AND, || - OR
- Truth Table

not

x	!x
false	true
true	false

and

x	y	x&&y
false	false	false
false	true	false
true	false	false
true	true	true

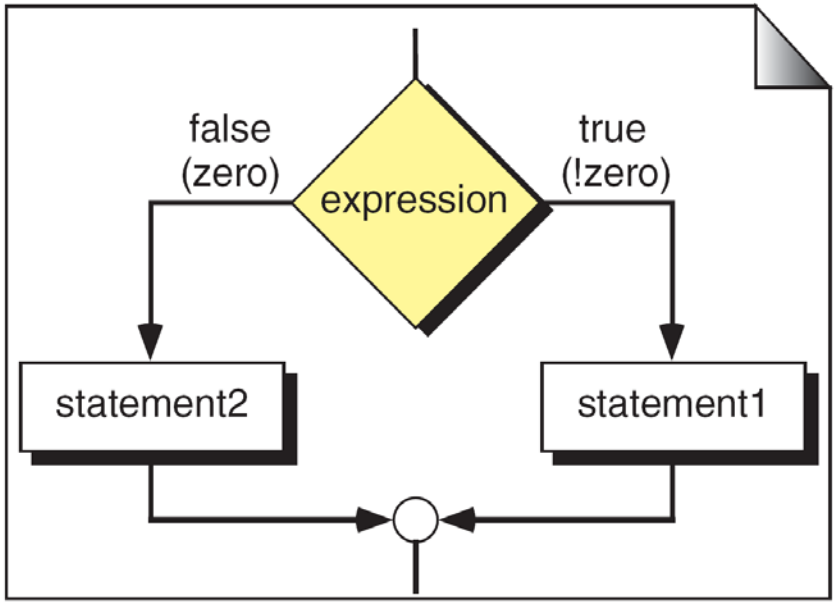
or

x	y	x||y
false	false	false
false	true	true
true	false	true
true	true	true

- 괄호를 잘 쓰자
  - a가 1~100까지의 짝수
  - if(a>=1 && a<=100 && a%2 == 0) → if((a>=1 && a<=100) && a%2 == 0)

2-Way Selection – if ~ else

```
if (조건)
{
        //조건이 참(True)일 때
}
else
{
        //조건이 거짓(False)일 때
}
```



(a) Logical Flow

```
if (expression)
    statement1
else
    statement2
```

(b) Code

if ~ else 특징

If 문만 만들 수도 있다. (else 가 필요 없을 시)

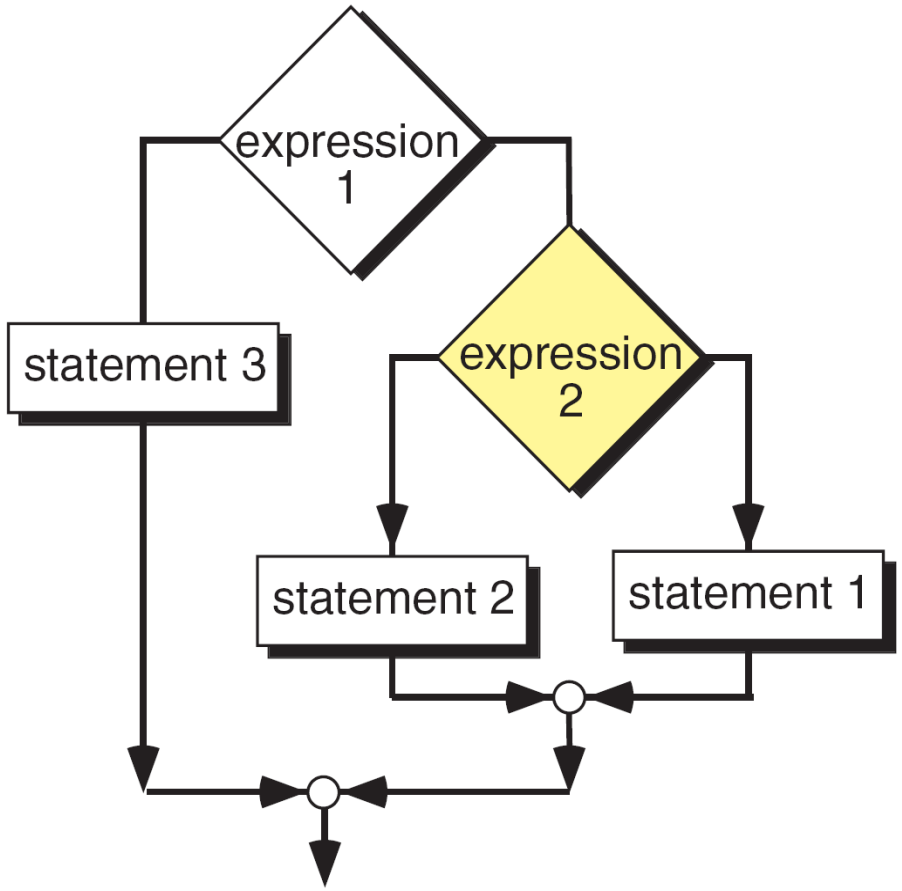
```
if ( 조건문 )
{
        //코드 입력
}
```

if ~ else example

```
int x = 1;
int y = 0;

if (x || y) {
    printf("x||y is true₩n");
}
else {
    printf("x||y is false₩n");
}
```

- 괄호 안 조건문의 참/거짓에 따라 실행할 부분을 선택
- 다른 부분은 실행되지 않고 넘어간다.

Nested *if* Statements



(a) Logic flow

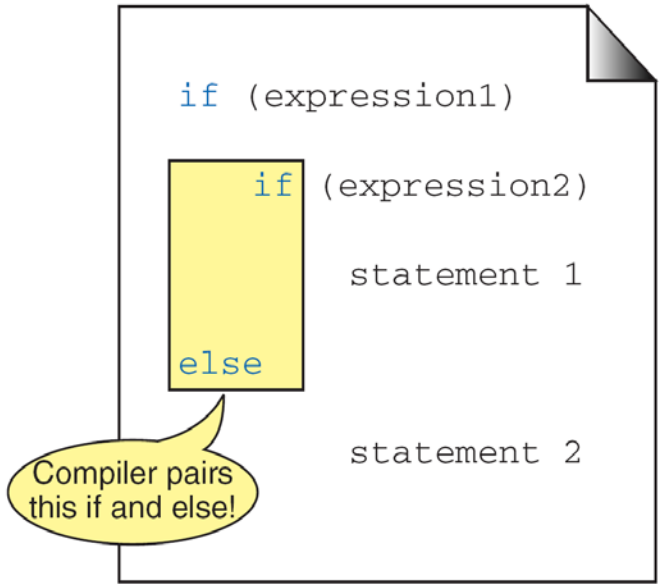
```
if (expression 1)
    if (expression 2)
        statement 1
    else
        statement 2
else
    statement 3
```

(b) Code

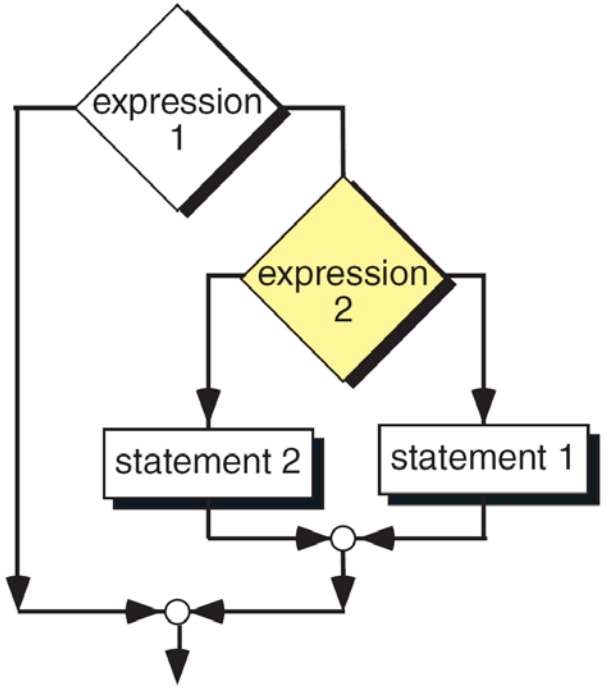
Nested if ~ else 주의 사항

- if, else다음에 실행할 명령이 하나이면 중괄호 필요 없음. But…

```
if (condition1)
        if(condition2)
                printf("Hello");

else
                printf("Hi");
```



(a) Code



(b) Logic Flow

***else* is always paired with the most recent unpaired *if*.**

Multi-Way Selection

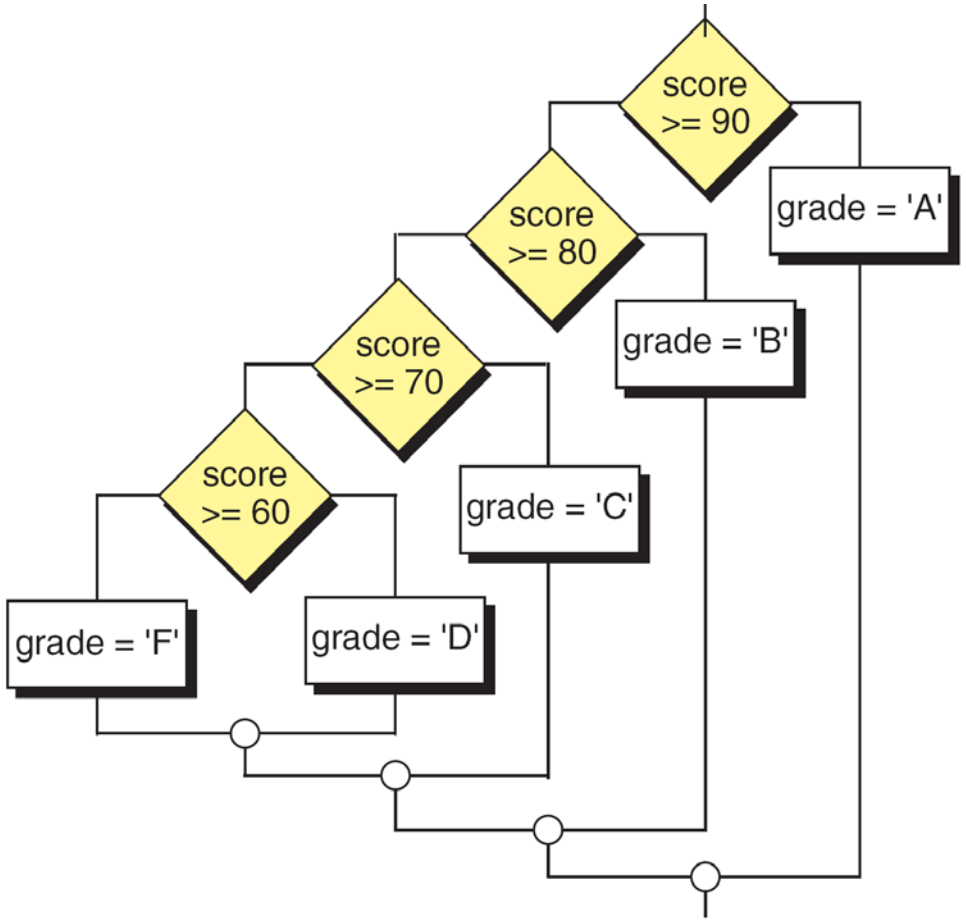
- 여러 가지 선택지 중에서 하나 (혹은 그 이상)을 선택해야 할 때

- 종류
  - else if 구문
  - switch 구문

if ~ else if ~ else

- if…else… 구문의 확장형

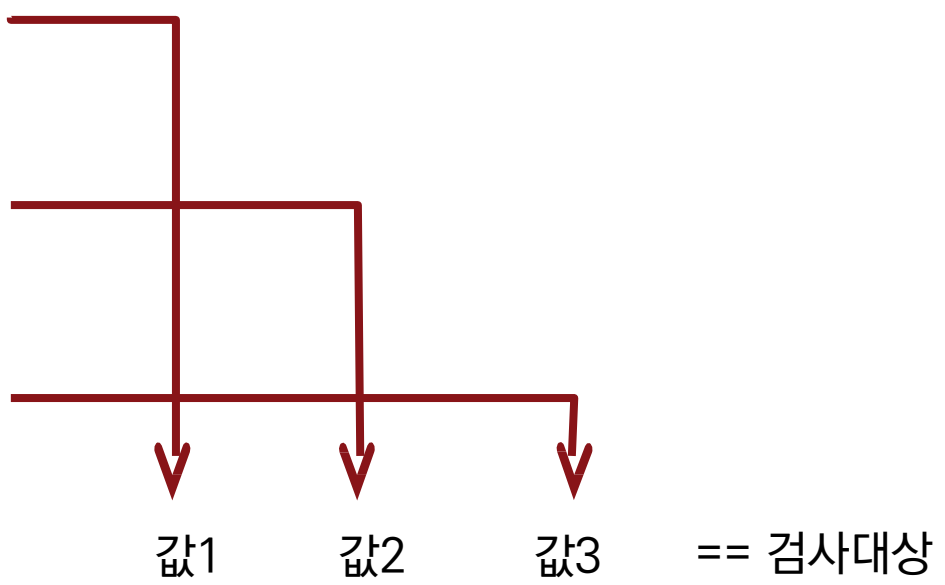
```
if( 조건문 )
{
        //조건문1 이 참일 때
}
else if(조건문2)
{
        //조건문1 이 거짓이고 조건문2가 참일 때
}
else
{
        //앞의 모든 조건문이 거짓일 때
}
```

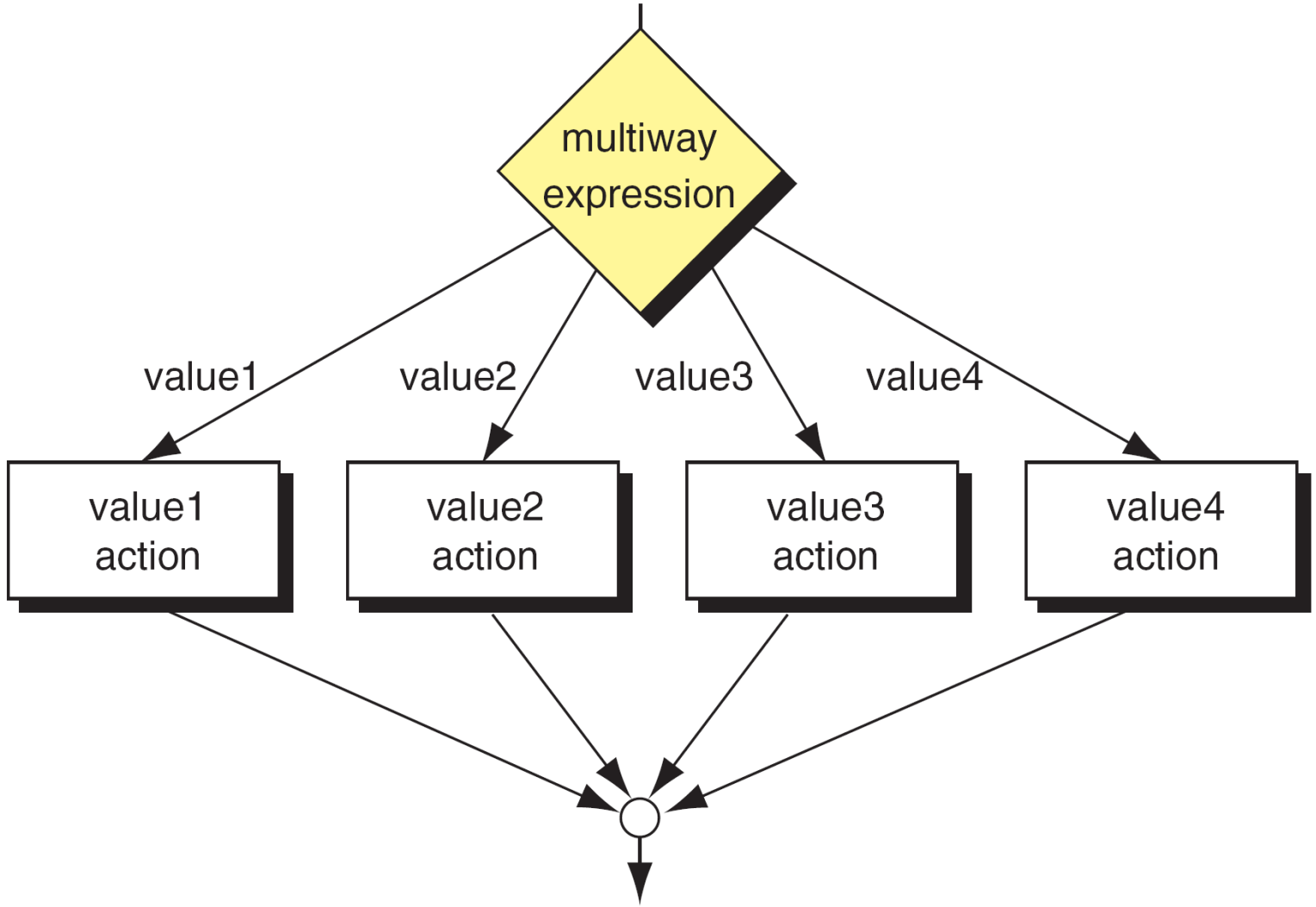


switch 문

- 검사하고자 하는 대상의 값(변수, 함수의 리턴 값)에 여러 가지 선택지가 있을 때

```
switch (검사대상)
{
      case 값1:
            statement1_1;
            statement1_2;
      case 값2 :
            statement2_1;
            statement2_2;
      case 값3 :
            statement3_1;
            statement3_2;
}
```




multiway
expression
value1
value2
value3
value4
value1
action
value2
action
value3
action
value4
action

break;

- switch, while, for, do while 같은 조건문 & 반복문을 빠져나가는 명령어
- if 문은 안됨
- 가장 가까운 하나만 빠져나갈 수 있다.
- 사용 용도
  - 특정한 순간에 조건문을 나가고 싶다.
  - 특정한 순간에 반복문을 나가고 싶다.

switch 문

```
switch(num) {
case 0:
        statement0;
        break;

case 1:
        statement1;
        break;
...
default:
        statementN;
        break;
}
```

- num에는 int형 정수
- num이 0이면 statement0 실행
- num이 1이면 statement1 실행
- num이 n이면 case n에 있는 statement 실행
- 만약 해당하는 case가 없을 경우에는 default에 있는 statement 실행

switch 문 주의사항

```
switch(num) {
case 0:
        statement0;

case 1:
        statement1;
        break;
...
default:
        statementN;
        break;
}
```

- case 0에 break;를 안 쓰면 어떻게 될까?
- num이 0 – case 0의 statement0가 실행될 때 break가 없어 case 1의 statement 1까지 실행된다!
- 의도한 경우가 아니라면 case마다 break;를 꼭 쓰자!

문제

- 학점 계산기 만들기
  - if~else문을 이용해서
  - switch문을 이용해서

- 60점 미만이면 F
- 60점 이상 70점 미만 D
- 70점 이상과 80점 미만 C
- 80점 이상 90점 미만 B
- 90점 이상 A

학점계산기 만들기 (if~else)

```
#include <stdio.h>

int main(void) {
   int score;
   printf("Type your score.₩n");
   scanf("%d", &score);

   if(score>=90)
            printf("Grade : A₩n");
   else if(score>=80)
            printf("Grade : B₩n");
   else if(score>=70)
            printf("Grade : C₩n");
   else
            printf("Grade : D₩n");
}
```

학점계산기 만들기 (switch)

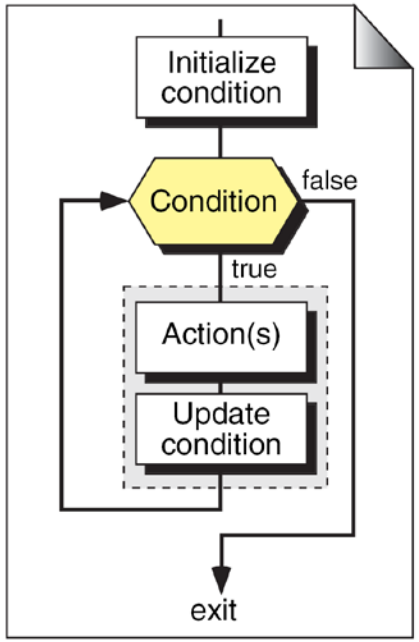
```
#include <stdio.h>

int main(void) {
        int score;
        printf("Input the score: ");
        scanf("%d", &score);
        switch(score/10) {
        case 9:
                printf("A grade₩n");
                break;
        case 8:
                printf("B grade₩n");
                break;
        case 7:
                printf("C grade₩n");
                break;
        case 6:
                printf("D grade₩n");
                break;
        default:
                printf("F grade₩n");
                break;
        }

        return 0;
}
```

반복문

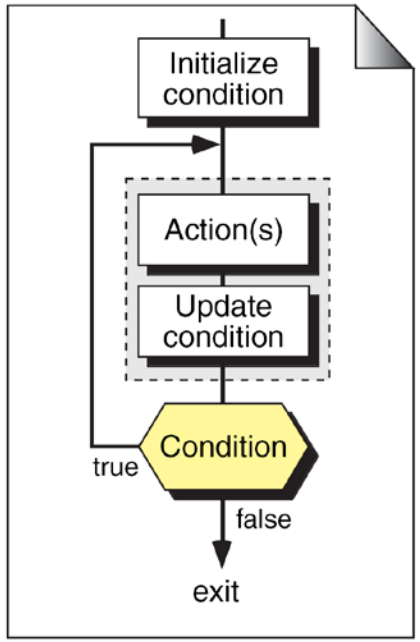
Loop (고리)

- Loop 문은 일련의 작업들을 반복해서 실행하게 하는 것.
  ex.1~50 까지의 합 구하기, 자동 냉방 장치 시스템
- pre-test loop: 검사하고 실행하기
  while 문, for 문
- post-test loop: 실행하고 검사하기
  do~while 문

- Requirement
  - Initialization
  - Update
  - Condition Check



(a) Pretest Loop



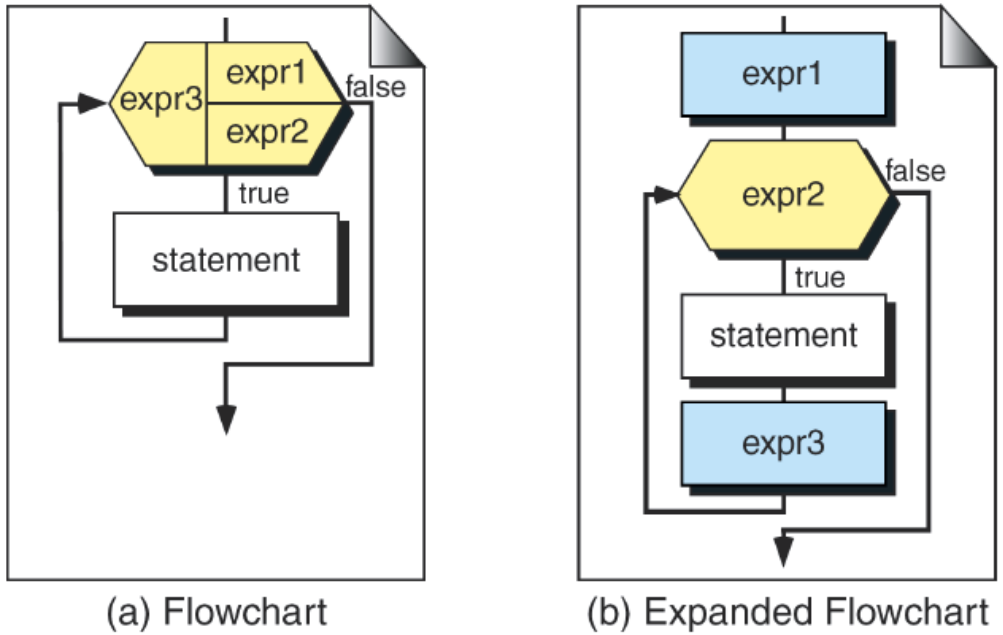
(b) Post-test Loop

for 문

• Pre-test Loop

```
for (initialization; condition test; update)
{
        //statement;
        //statement;
        …
}
```

I –〉 C (T) –〉 S –〉 U –〉 C (T) –〉 S –〉 U
–〉 C (T) –〉 S –〉 U –〉 C (F)–〉 loop out



(a) Flowchart (b) Expanded Flowchart

```
for (expr1; expr2; expr3)
    statement
```

예제

```
int i;
for(i=0; i<10; i++)
{
        printf("%d₩n",i);
}
```

결과 :

0 1 2 3 4 5 6 7 8 9

for문 문제

- 1~100 사이의 수 중 5의 배수 갯수세기

```
#include <stdio.h>

main(void) {
   int i, cnt=0;

   for(i=1; i<=100; i++) {
                if (i%5==0)
                              cnt++;

   }
   printf("%d₩n", cnt);
}
```

for문 구구단

- 원하는 단 입력 받아 구구단 출력하기

```
#include <stdio.h>
main()
{
        int i, num;
        scanf("%d", &num);
        for(i=1; i<=9; i++) {
                printf("%d * %d = %d₩n", num, i, num*i);
        }
}
```

Nested for loop

```
 5  #include <stdio.h>
 6
 7  int main (void)
 8  {
 9  // Statements
10     for (int i = 1; i <= 3; i++)
11        {
12         printf("Row %d: ", i);
13         for (int j = 1; j<= 5; j++)
14              printf("%3d", j);
15         printf("\n");
16        } // for i
17     return 0;
18  }  // main
```

```
Results:
Row 1:    1   2   3   4   5
Row 2:    1   2   3   4   5
Row 3:    1   2   3   4   5
```

다중 for문 문제

- 구구단 전체 출력하기

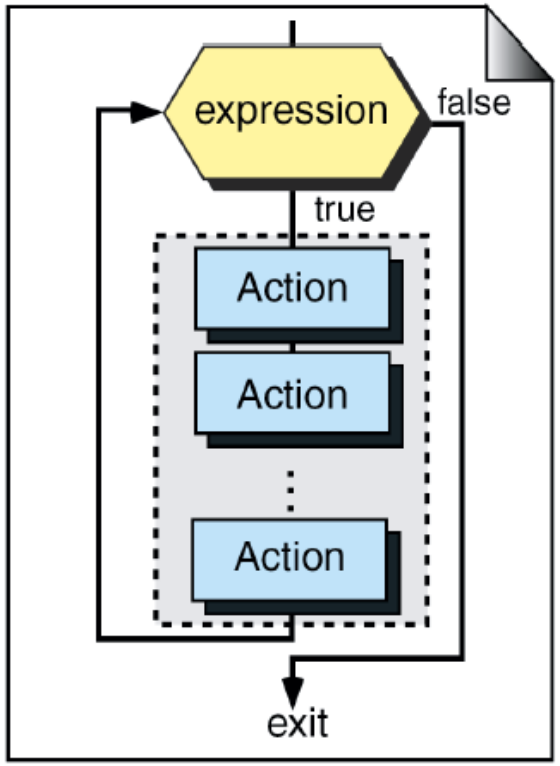
```
#include <stdio.h>
main() {
	int i, j;
	for(i=2; i<=9; i++) {
		for(j=1; j<=9; j++) {
			printf("%d * %d = %d₩n", i, j, i*j);
		}
	}
	printf("₩n");
}
```

while 문

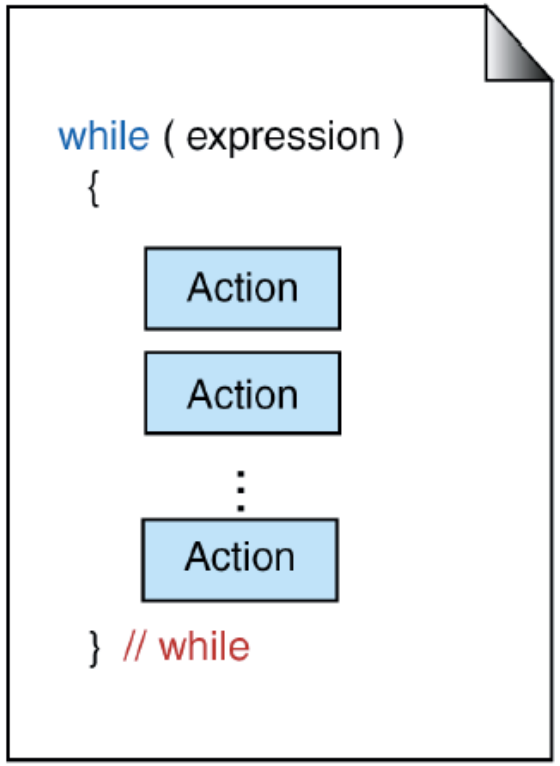
- Pre-test Loop
- 조건이 True인 동안 Statements를 계속 실행

```
while (condition)
{
    //statement;
    //statement;;
        ...
}
```

C -> S -> C -> S -> C -> loop out
T T F



(a) Flowchart



(b) C Language

while 문을 for문 처럼

```
int i;
I = 0;
while(i 〈10)
{
    printf(“%d₩n”);
    i++;
}
```

while문 구구단

- 원하는 단 입력받아 구구단 출력하기

```
#include <stdio.h>
main() {
        int i=1, num;
        scanf("%d", &num);
        while(i <= 9)    // for(i=1; i<=9; i++) {
                printf("%d * %d = %d₩n", num, i, num*i);
                i++;
        }
}
```

while문

- 정수를 입력받아 더해 나가다가 0을 입력하면 입력된 모든 정수의 합을 출력하고 프로그램 종료시키기

```
#include <stdio.h>
main()
{
        int sum=0;
        int su=1;
        while(su!=0)
        {
            scanf("%d",&su);
            sum += su;
        }
        printf("sum=%d₩n", sum);
}
```

이중 while문

- 구구단 전체 출력하기

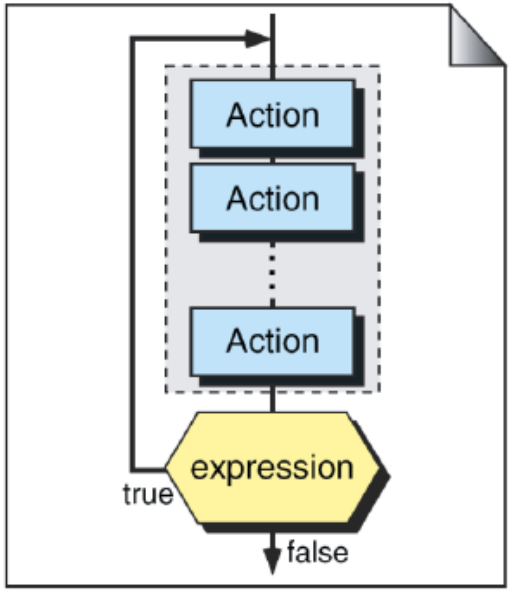
```
#include <stdio.h>

int main(void) {
    int dan=2;
    int i;
    while(dan<10) {
            i=1;            // 새로운 단의 시작을 위해서
            while(i < 10) {
                printf("%d × %d = %d ₩n", dan, i, dan*i);
                i++;
            }
            dan++;      // 다음 단으로 넘어가기 위한 증가
            printf("₩n");
    }
}
```

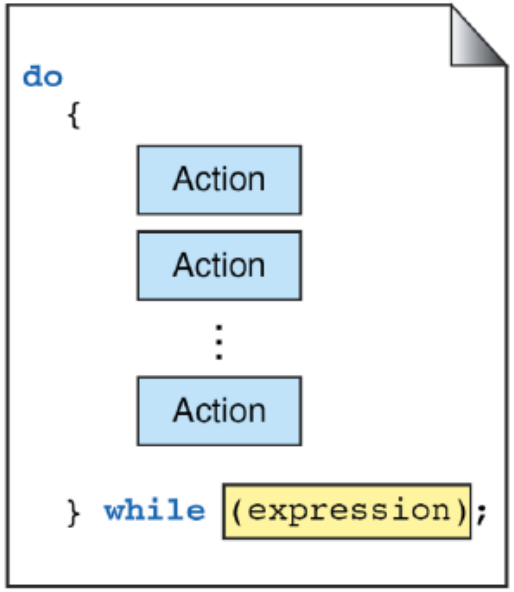
do~while 문

- Post-test Loop
- 조건문이 만족할 동안 Statements를 실행
- 최초 1번은 반드시 실행됨

```
do
{
    //statement;
    //statement;
        ...
} while(condition);
```

S-> C -> S -> C -> S -> C -> loop out





while문과 do~while 문의 차이

```
int main(void) {
    int sum = 0;
    int i = 6;

    do {
        sum += i;
        i++;
    } while (i <= 5);

    printf("sum = %d\n", sum);
    return 0;
}
```

Result:
sum = 6

```
int main(void) {
    int sum = 0;
    int i = 6;

    while (i <= 5) {
        sum += i;
        i++;
    }
    printf("sum = %d\n", sum);
    return 0;
}
```

Result:
sum = 0

do~while문 예제

- 정수를 입력받아 더해 나가다가 0을 입력하면 입력된 모든 정수의 합을 출력하고 프로그램 종료시키기

```
#include <stdio.h>
main() {
        int sum=0;
        int num=0;
        do {
            scanf("%d",&num);
            sum += num;
        } while(num!=0)
        printf("sum=%d₩n", sum);
}
```

Requirement가 필요 없는 경우

- Requirement들이 필요에 따라 없을 수도 있다.
- Initialization 없는 경우
  반복문을 위한 특별한 변수가 필요 없을 때

- Condition 없는 경우
  그냥 무조건 참. 무한 루프 ex) while()

- Update 없는 경우
  음…Think…

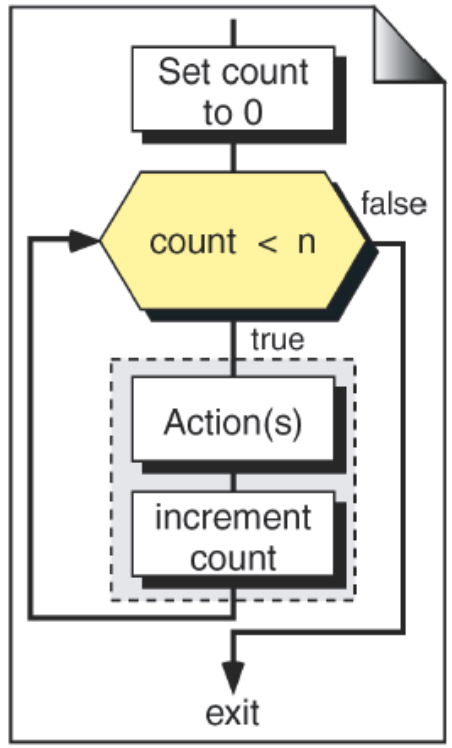
for and *while* as Perpetual Loops

```
1  // A bad loop style
2  for ( ; ; )
3     {
4      …
5      if (condition)
6         break;
7     } // for
```

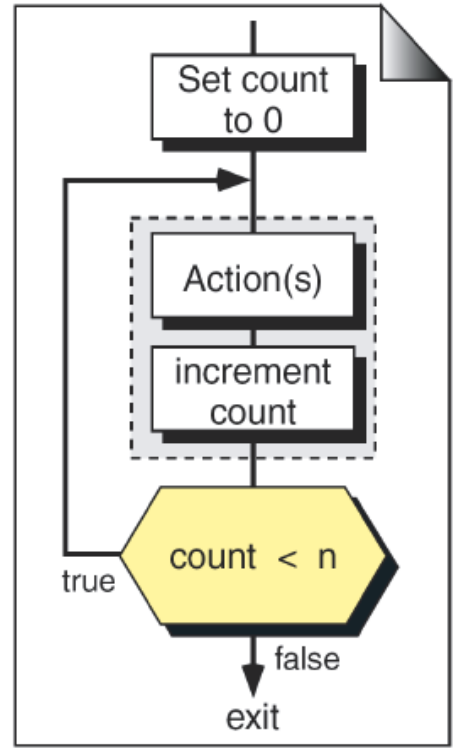
```
1  // A better loop style
2  for ( ; !condition ; )
3     {
4       …
5     } // for
```

```
1  while (x)
2     {
3      …
4      if (condition)
5         break;
6      else
7         …
8     } // while
```

```
1  while  (x && !condition)
2     {
3      …
4      if (!condition)
5           …;
6     } // while
```



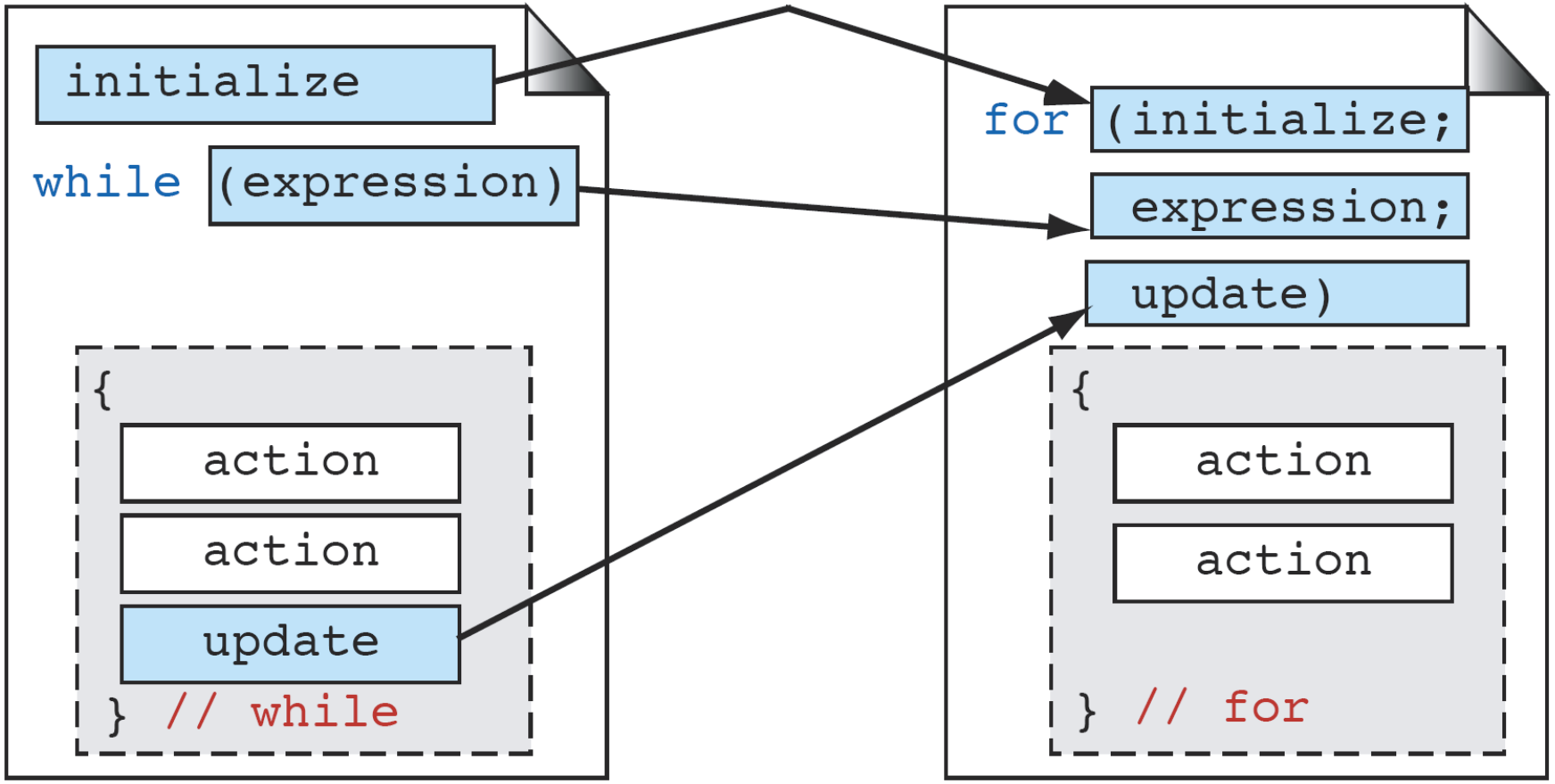
(a) Pretest Loop



(b) Post-test Loop

Pretest Loop		Post-test Loop	
Initialization:	1	Initialization:	1
Number of tests:	*n* + 1	Number of tests:	n
Action executed:	n	Action executed:	n
Updating executed:	n	Updating executed:	n
Minimum iterations:	0	Minimum iterations:	1

for문과 while문 비교



언제 어떤걸 쓰나요

- for: loop 를 몇번 돌아야 하는 지 알고있는 경우 주로 사용
  → “**회만 돌려야겠다.”
- While, do~while: 몇번 돌아야 하는 지 모르는 경우 주로 사용
  → “조건 충족때 까지 돌리고 싶다!!”

- 조금만 고치면 세 개가 다 의미가 같도록 할 수 있기 때문에 사실 원하는 결과가 나오게만 잘 쓰면 되긴함.

for ↔ while ↔ do ~ while

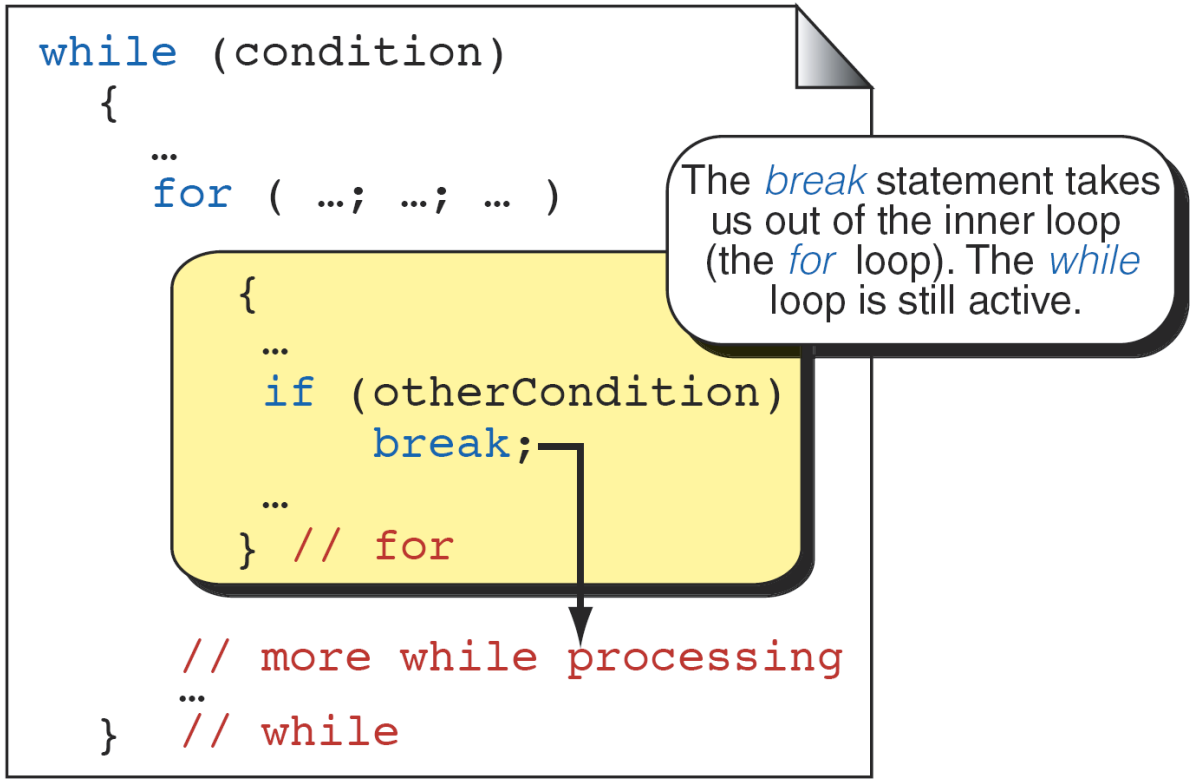
분기문

- Break
  - 가장 가까운 switch문, loop 문 한 개를 나간다.

- Continue
  - loop 문 안에 쓰여서, 바로 Condition 으로 간다.

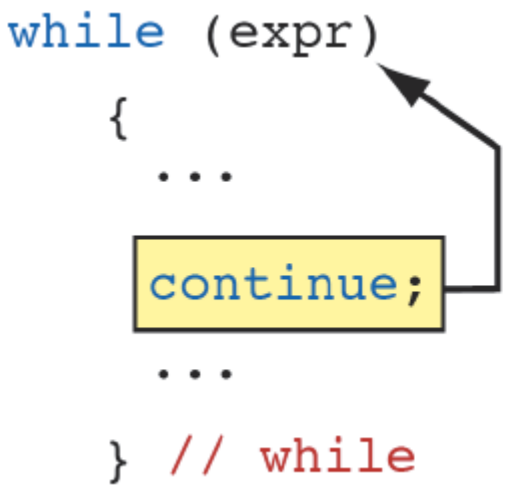
break ;

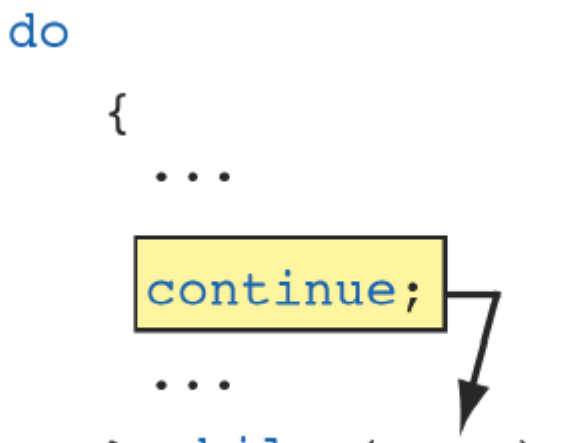
- 가장 가까운 loop 한 개를 빠져나간다.

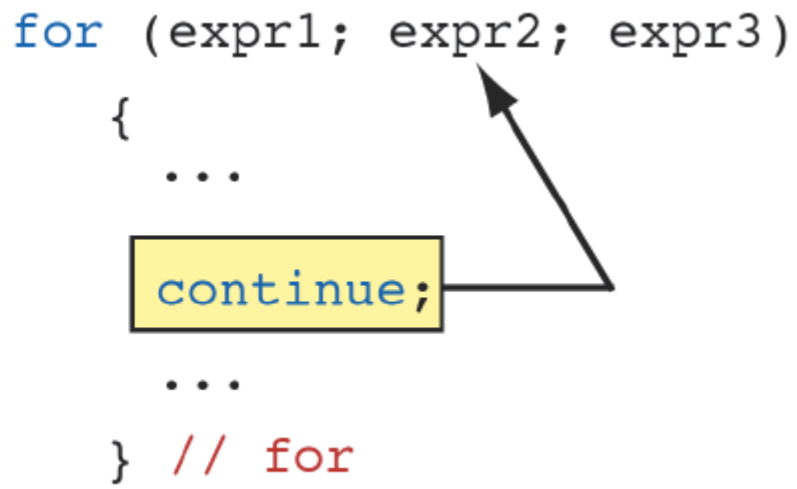


Continue

- for 문에서는 continue 를 만나면 update → condition check
- while, do~while에서는 continue 를 만나면 condition check만







The Comma Expression

- Complex expression made up of two expressions separated by a comma
- Most often used in for statements

```
for (sum = 0, i = 1; i <= 20; i++) {
                scanf("%d", &a);
                sum += a;
} // for
```

```
while (testCount++, loopCount <= 10)

do
   printf("%3d",    loopCount++);
while (testCount++,  loopCount <= 10);
```

마치기 전에

다음 시간

- Again, 함수(function)
- 재귀함수(recursion)
- Array + Searching