전산 SMP 4주차

2014. 10. 14

김범수

bskim45@gmail.com

지난시간 복습

조건문 (Selection Statement)

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

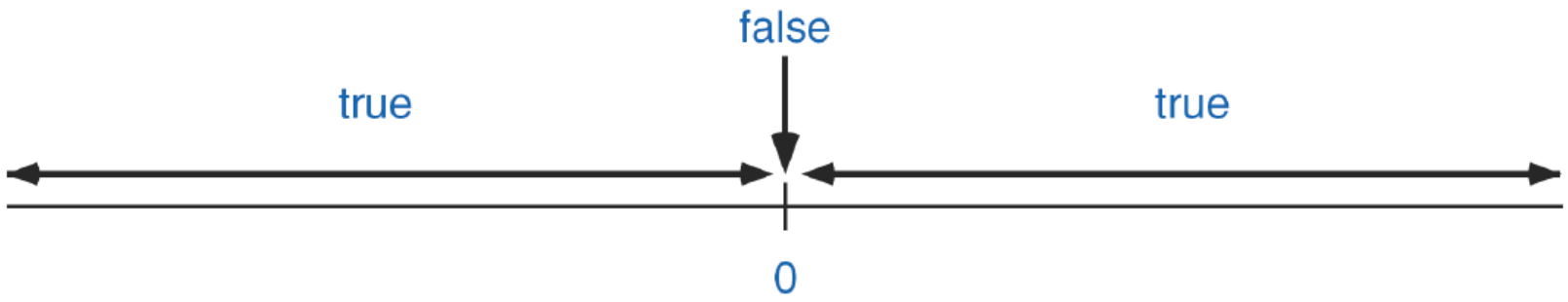
2-Way Selection
  - 두 가지 선택지 중에 한 가지

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

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

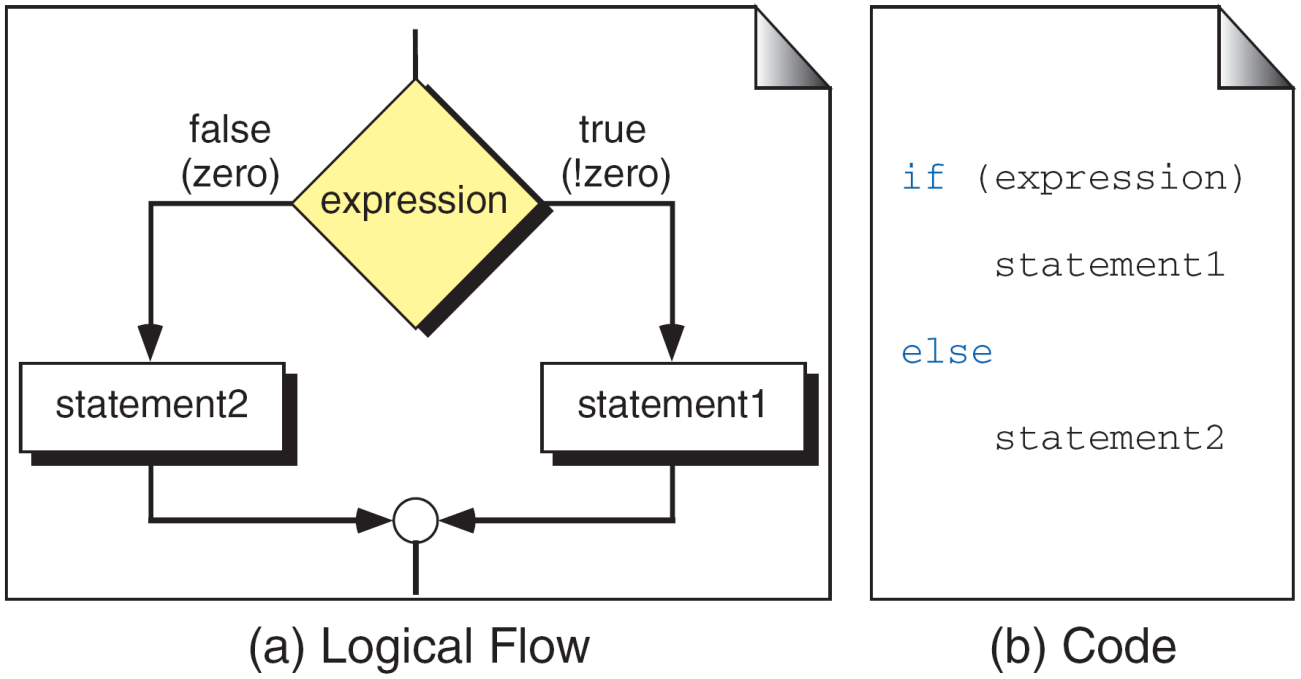
조건 Condition (TRUE / FALSE)

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



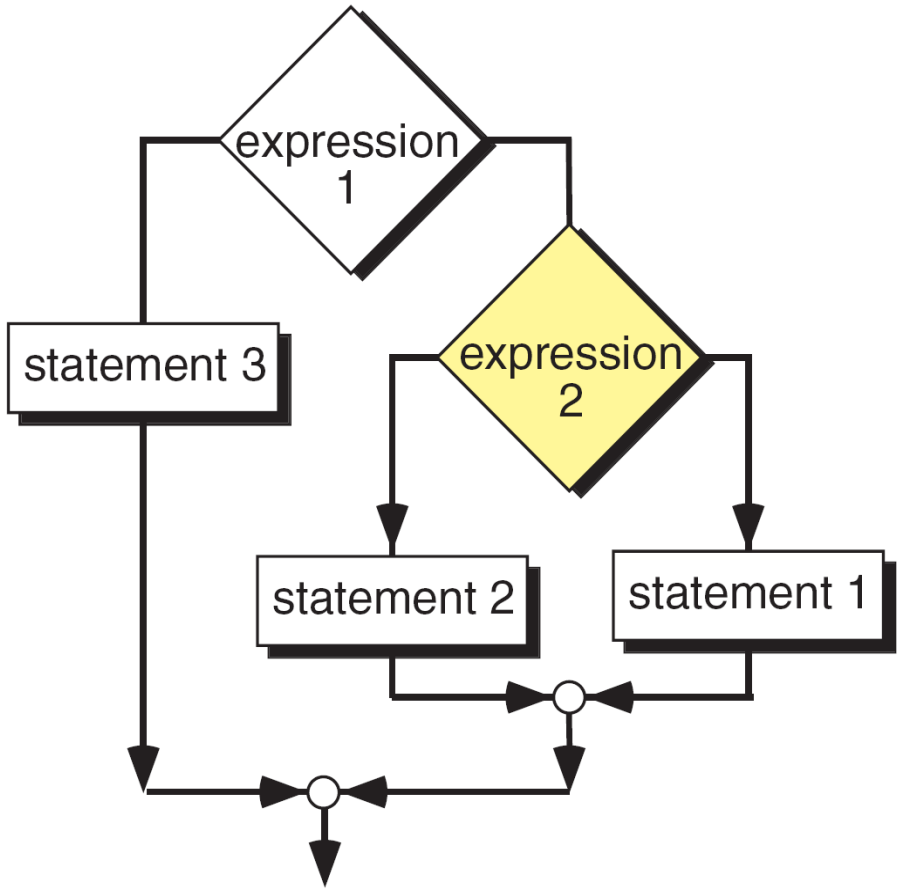
2-Way Selection – if ~ else

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



(a) Logical Flow (b) Code

Nested *if* Statements



(a) Logic flow

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

(b) Code

Multi-Way Selection

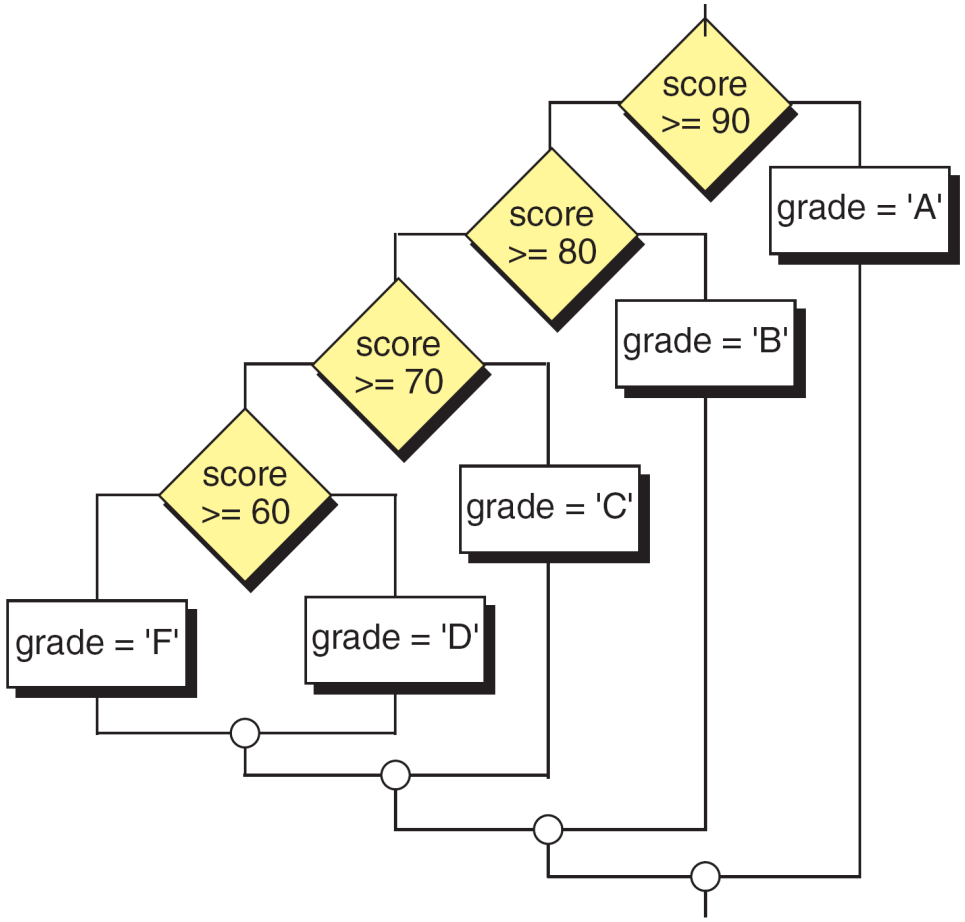
- 여러 가지 선택지 중에서 하나 (혹은 그 이상)을 선택해야 할 때

- 종류
  - else if 구문
  - switch 구문

if ~ else if ~ else

- if…else… 구문의 확장형

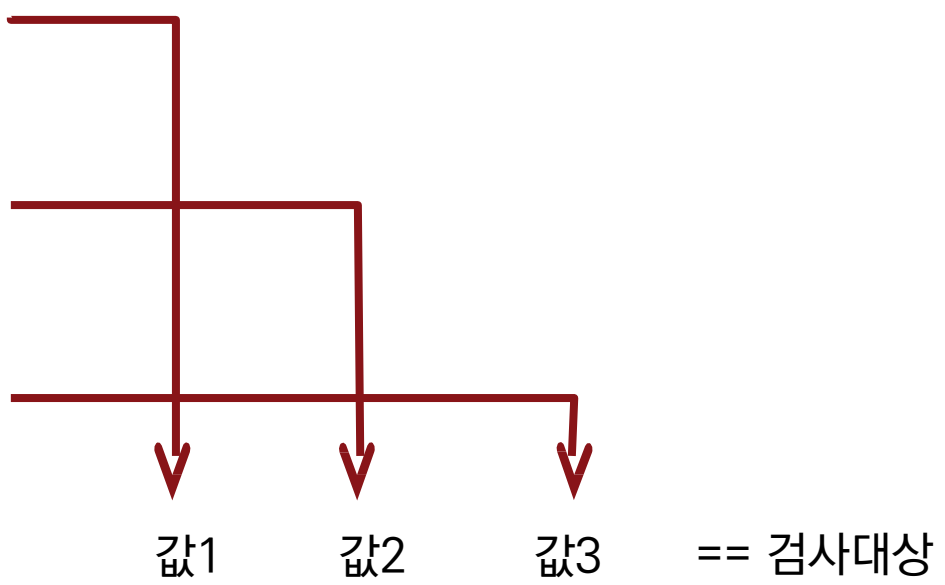
```
if( 조건문 )
{
        //조건문1 이 참일 때
}
else if(조건문2)
{
        //조건문1 이 거짓이고 조건문2가 참일 때
}
else
{
        //앞의 모든 조건문이 거짓일 때
}
```



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;
}
```



break;

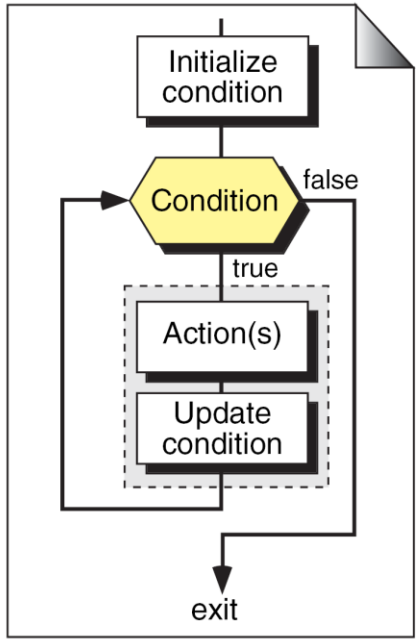
- switch, while, for, do while 같은 조건문 & 반복문을 빠져나가는 명령어
- if 문은 안됨
- 가장 가까운 하나만 빠져나갈 수 있다.
- 사용 용도
  - 특정한 순간에 조건문을 나가고 싶다.
  - 특정한 순간에 반복문을 나가고 싶다.

반복문

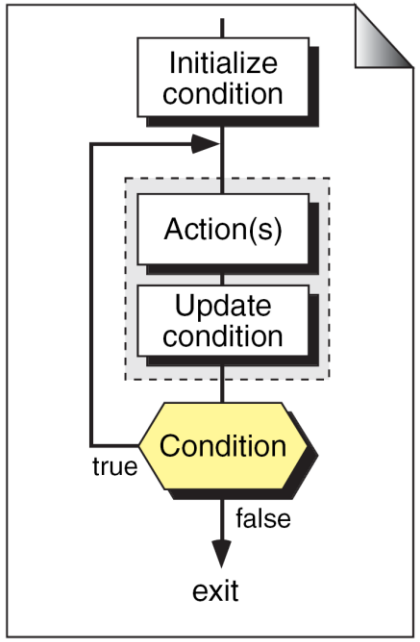
Loop (고리)

- 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



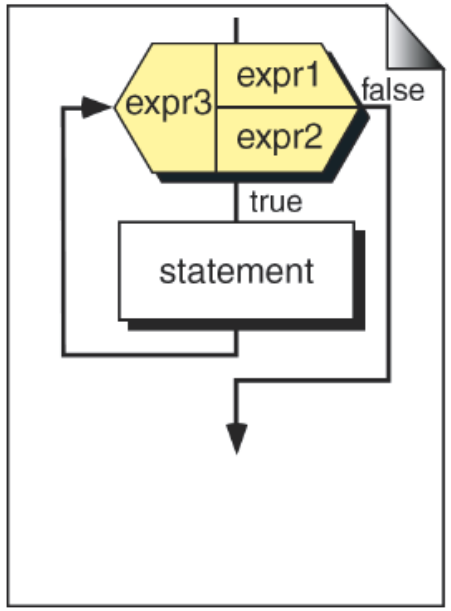
(b) Post-test Loop

for 문

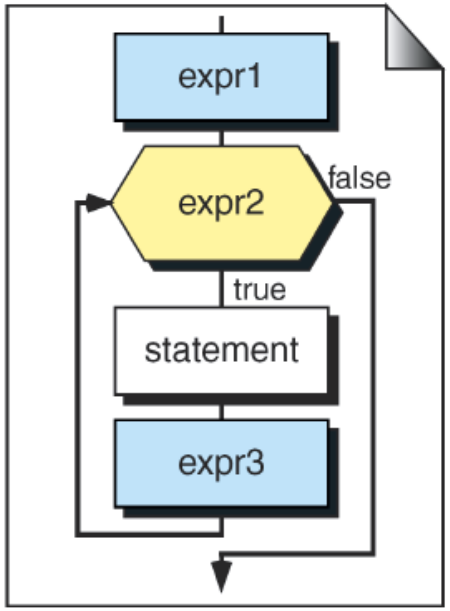
• Pre-test Loop

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

I -> S -> U -> C (T) -> S -> U -> C (T)
-> S -> U -> C (F)-> loop out



(a) Flowchart



(b) Expanded Flowchart

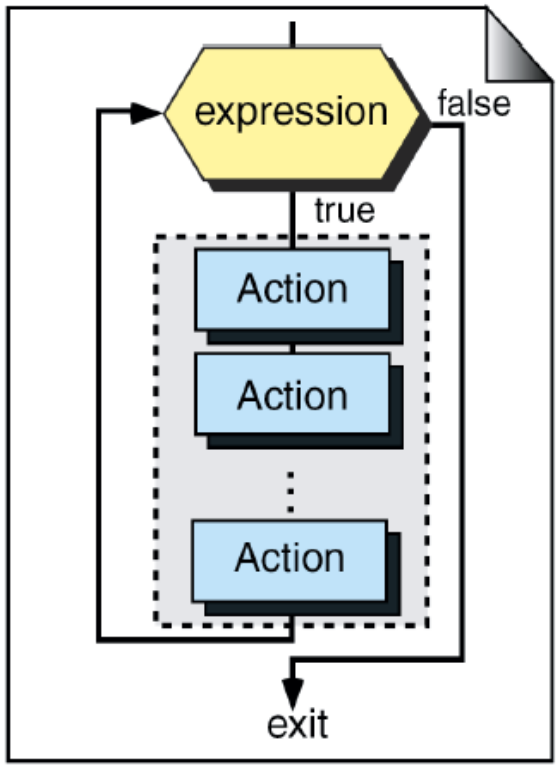
```
for (expr1; expr2; expr3)
    statement
```

while 문

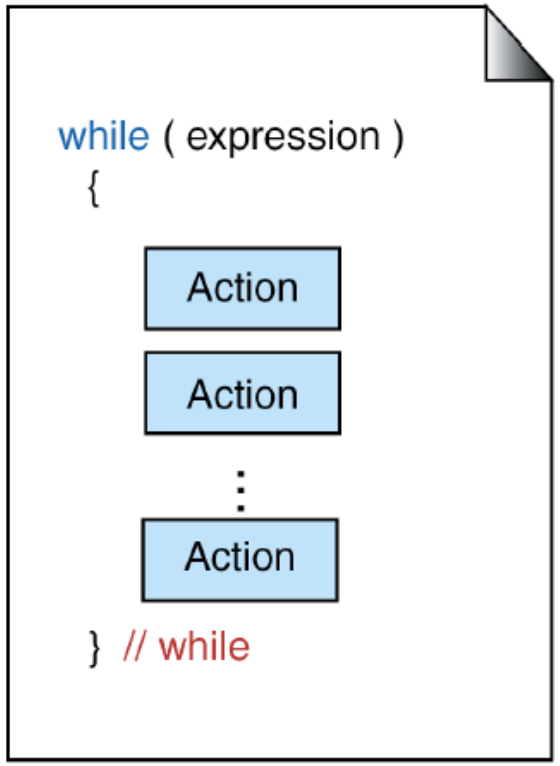
- Pre-test Loop
- 조건이 True인 동안 Statements를 계속 실행

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

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



(a) Flowchart



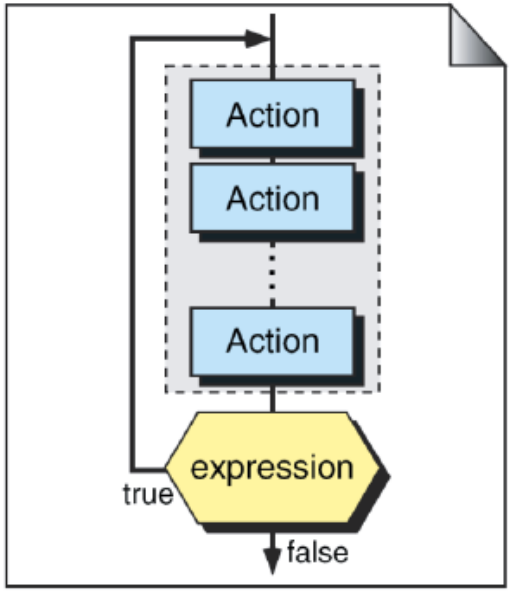
(b) C Language

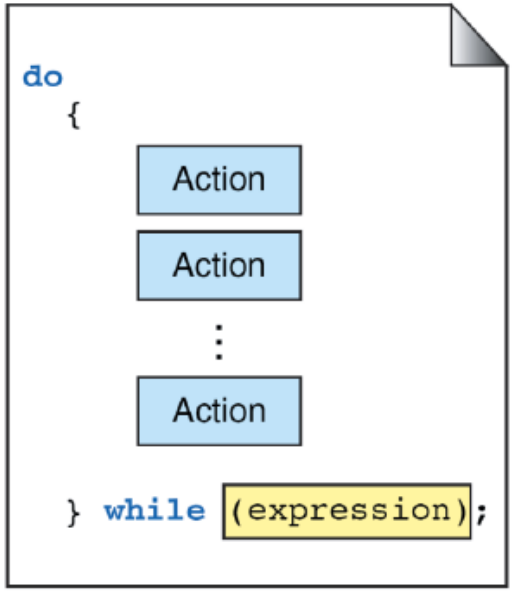
do~while 문

- 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





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…

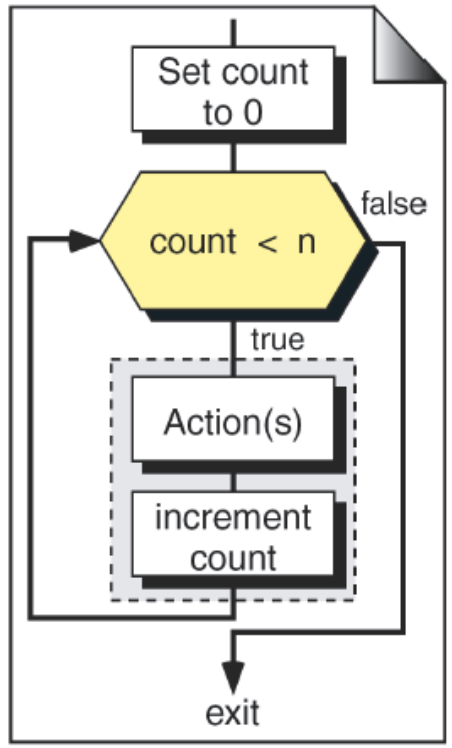
for and *while* as Perpetual Loops

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

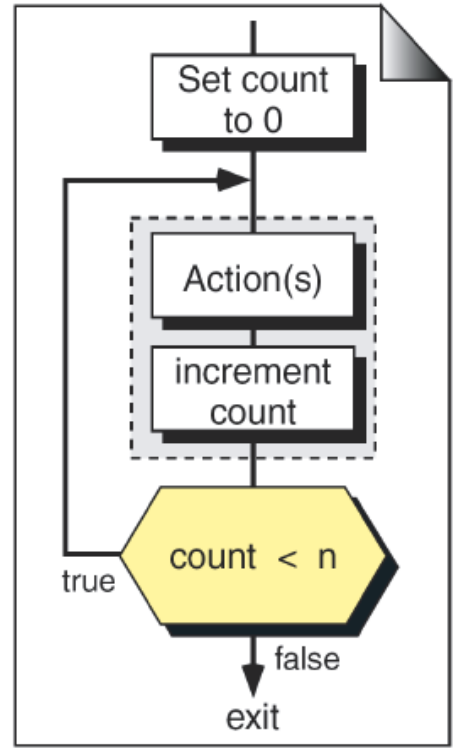
```
1  // A better loop style
2  for ( ; !condition ; )
3     {
4       …
5     } // for
```

```
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
```



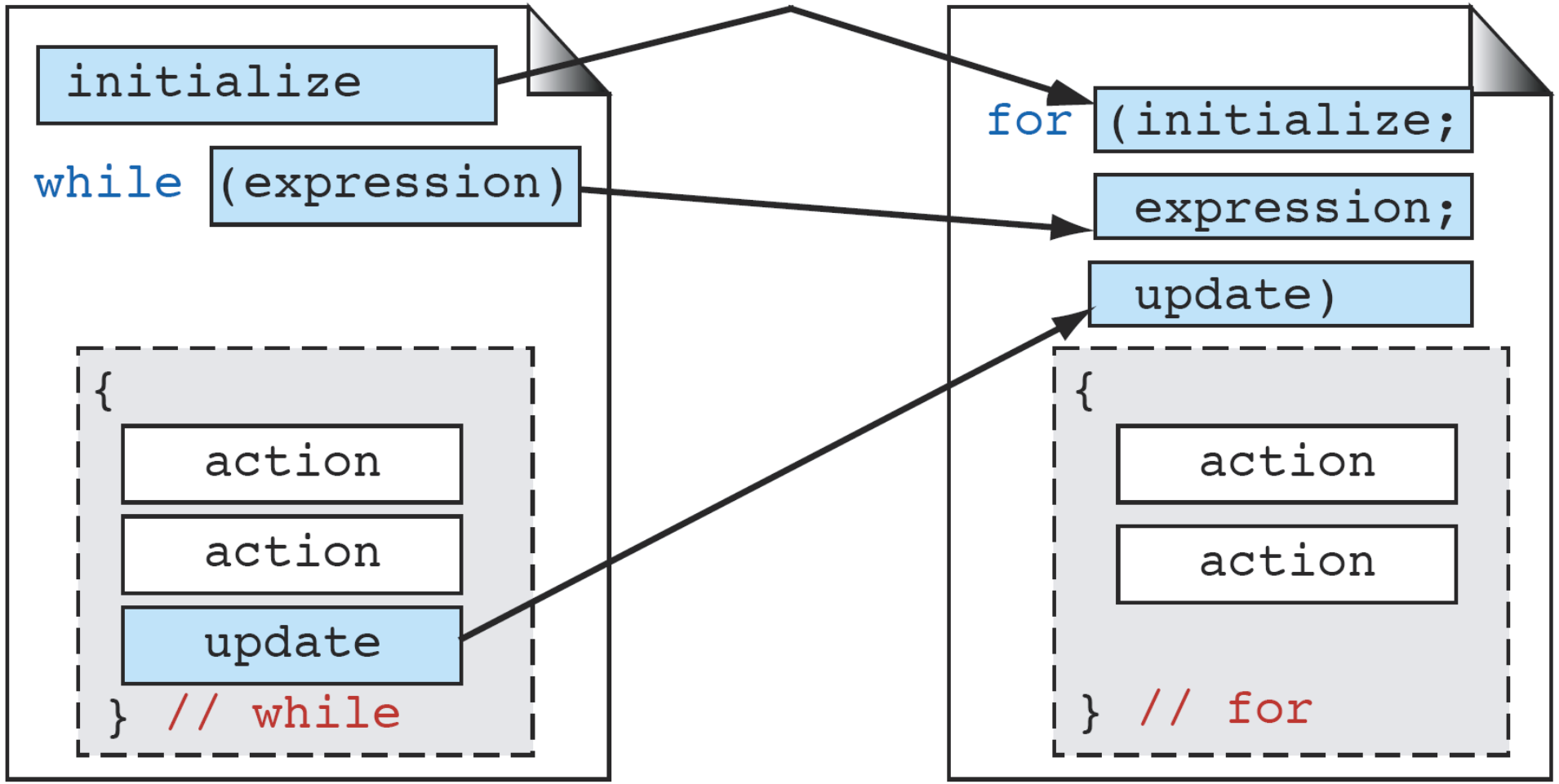
(a) Pretest Loop



(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 충족 때 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

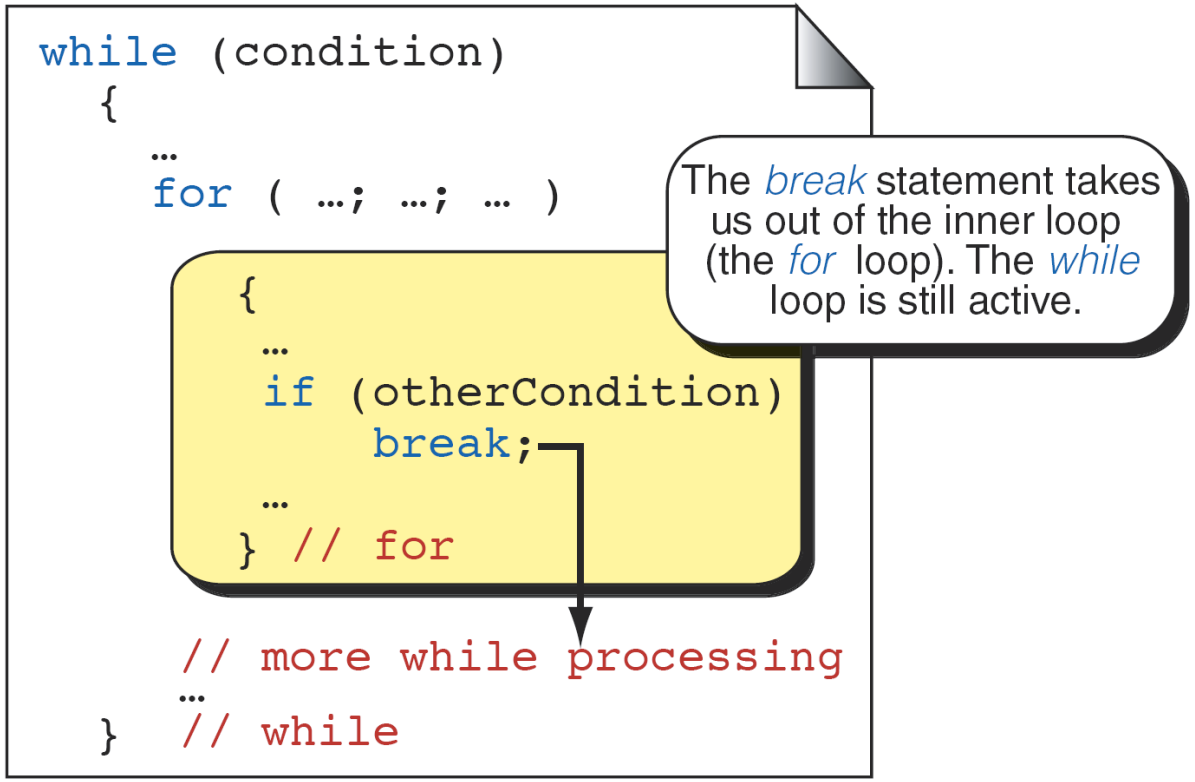
분기문

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

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

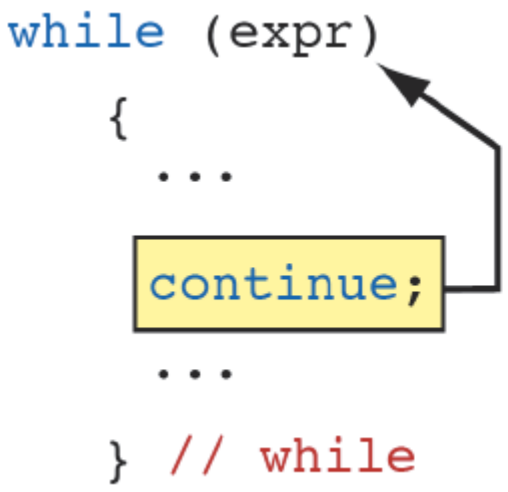
Break

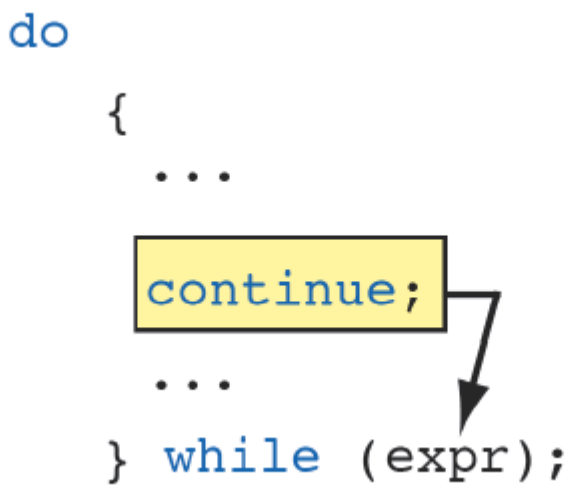
- 가장 가까운 loop 한 개를 빠져나간다.

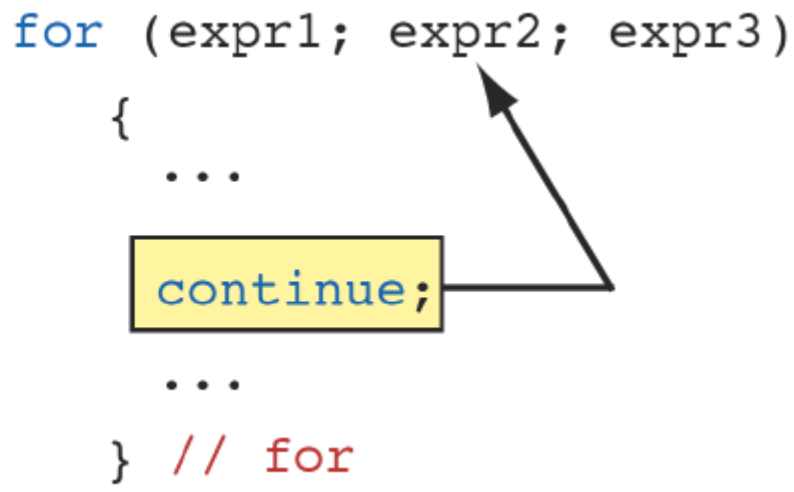


Continue

- for 문에서는 Continue 를 만나면 update -〉 condtion을 거친다.
- while, do~while에서는 Continue 를 만나면 condition만 거친다. 그 것밖에 없기도 하고…







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);
```

반복문 예제

모양 만들기

1)
O
OO
OOO
OOOO
OOOOO

2)
OOOOO
OOOO
OOO
OO
O

3)
O
OO
OOO
OOOO
OOOOO

4)
OOOOO
OOOO
OOO
OO
O

5)
O
OOO
OOOOO

6)
OOOOO
OOO
O

모양 만들기 1

```
#include <stdio.h>
int main(void) {
    int i=0, j=0, k=5;
    while(i<5) {
        while(j<=i) {
            printf("O");
            j++;
        }
        printf("\n");
        i++;
        j=0;
    }
    return 0;
}
```

1)
O
OO
OOO
OOOO
OOOOO

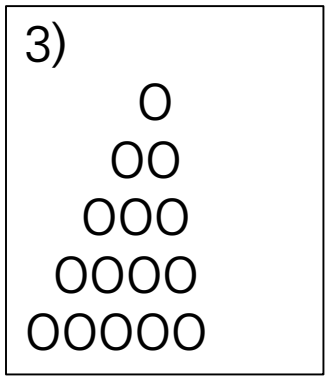
모양 만들기 2

```
#include <stdio.h>
void main() {
    int i=0, j=5, k=5;
    while(i<5) {
        while(j>i) {
            printf("O");
            j--;
        }
        printf("\n");
        i++;
        j=5;
    }
}
```

2)
OOOOO
OOOO
OOO
OO
O

모양 만들기 3

```
#include <stdio.h>
void main() {
    int i=0, j=4, k=0;
    while(i<5) {
            while(j>i) {
                    printf(" ");
                    j--;
            }
            while(k<=i) {
                    printf("O");
                    k++;
            }
            i++;   j=4;  k=0;
            printf("\n");
            }
}
```



모양 만들기 4

```
#include <stdio.h>
void main() {
    int i=0, j=5, k=5;
    while(i<5) {
        while(j<i) {
            printf(" ");
            j++;
        }
       while(k>i){
           printf("O");
           k--;
        }
        printf("\n");
        i++, j=0, k=5;
    }
}
```

4)
OOOOO
 OOOO
  OOO
   OO
    O

모양 만들기 5

```
#include <stdio.h>
void main() {
    int i=0, j, k=0, temp=0, row;
    scanf("%d",&row);
    while(i<row) {
       j=row;
        while( j-1>i) {
            printf(" ");
            j--;
        }
       while(k<=temp) {
          printf("O");
            k++;
        }
         printf("\n");
         i++, k=0, temp+=2;
    }
}
```

5)
3입력

O
OOO
OOOOO

모양 만들기 6

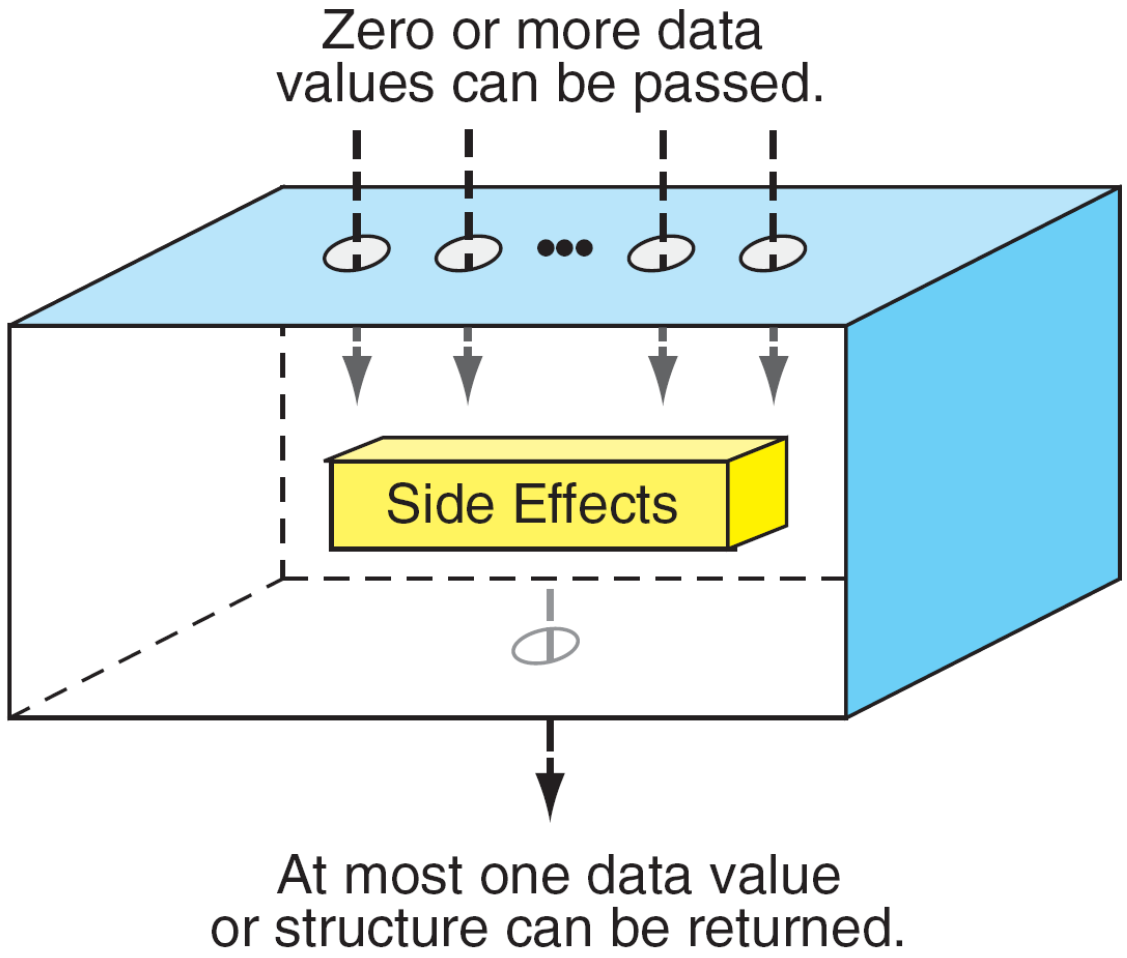
```
#include <stdio.h>
void main() {
    int i=0, j, k=0, temp=0, row;
    scanf("%d",&row);
    imsi = row*2-1;
    j = row-1;
     while(i<row) {
        while(i>j) {
             printf(" ");
             j++;
         }
         while(k<temp){
            printf("O");
             k++;
         }
         printf("\n");
         i++, k=0,j=0, temp-=2;
    }
}
```

6)
3입력

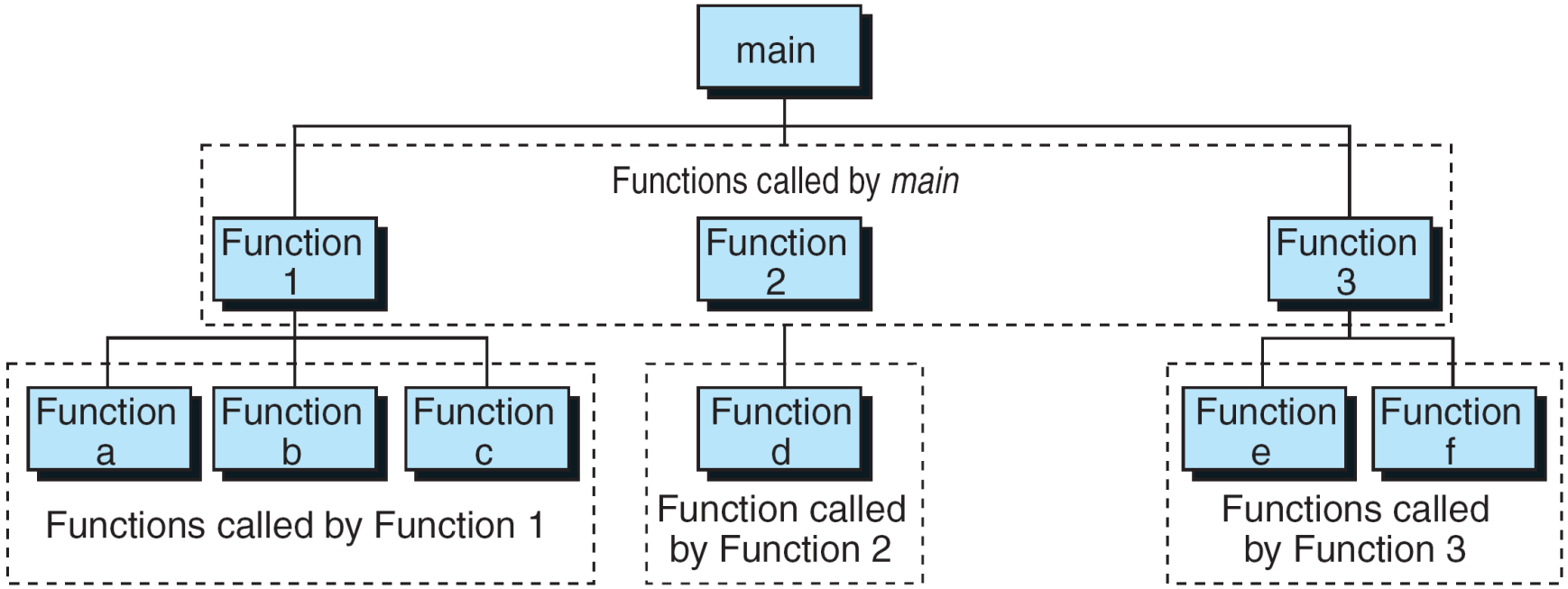
OOOOO
OOO
O

함수

함수

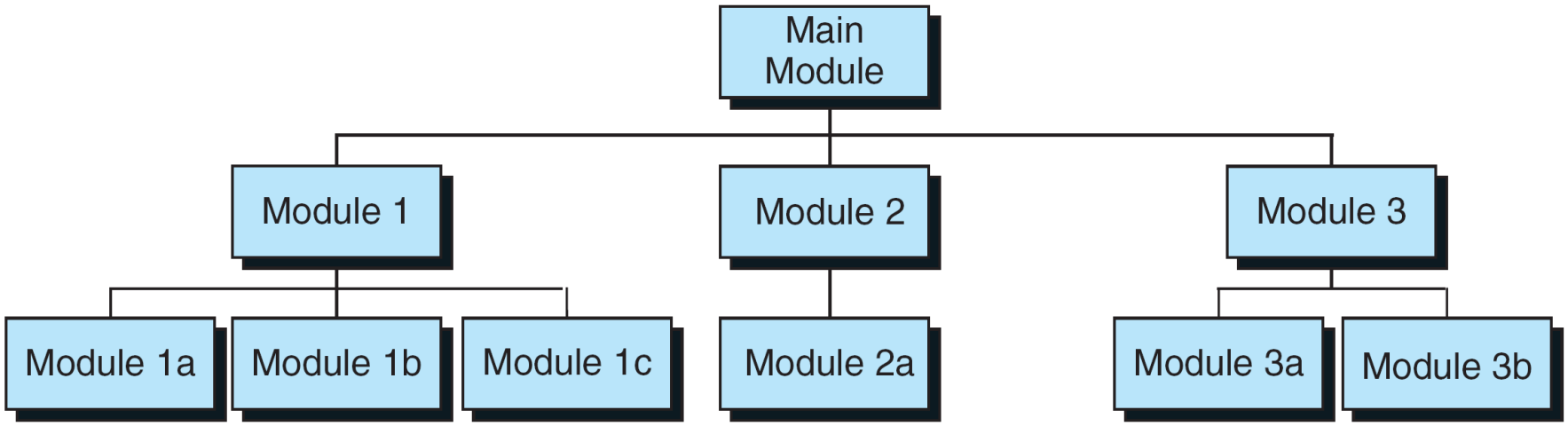

Zero or more data
values can be passed.
Side Effects
At most one data value
or structure can be returned.

C는 한 개 이상의 함수로 구성된다



함수를 왜 만들죠

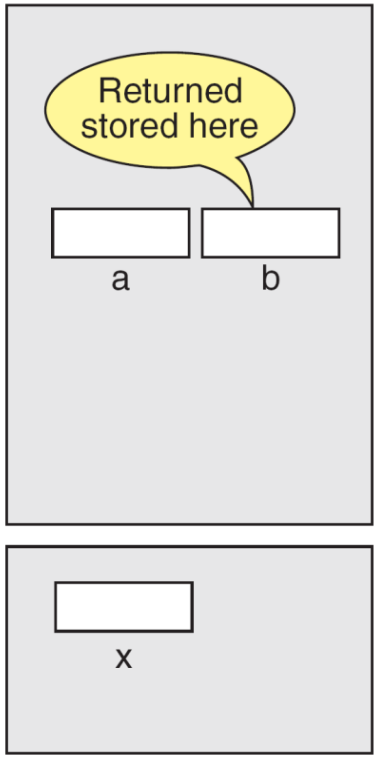
- 함수란 특정한 작업을 수행하도록 독립적으로 작성된 프로그램
- 프로그램에서 반복적으로 수행되는 부분을 함수로 작성하여 필요할 때마다 호출하여 사용
- 코드의 분할과 재사용



Calling a Function That Returns a Value

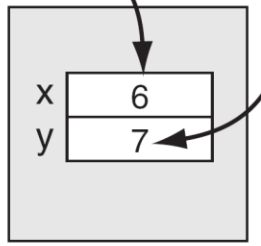
```c
// Function Declaration
int sqr (int x);
int main (void)
{
// Local Declarations
   int a;
   int b;
// Statements
   scanf("%d", &a);
   b  =  sqr (a);
   printf("%d squared: %d\n", a,  b);
   return 0;
}  // main
```

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



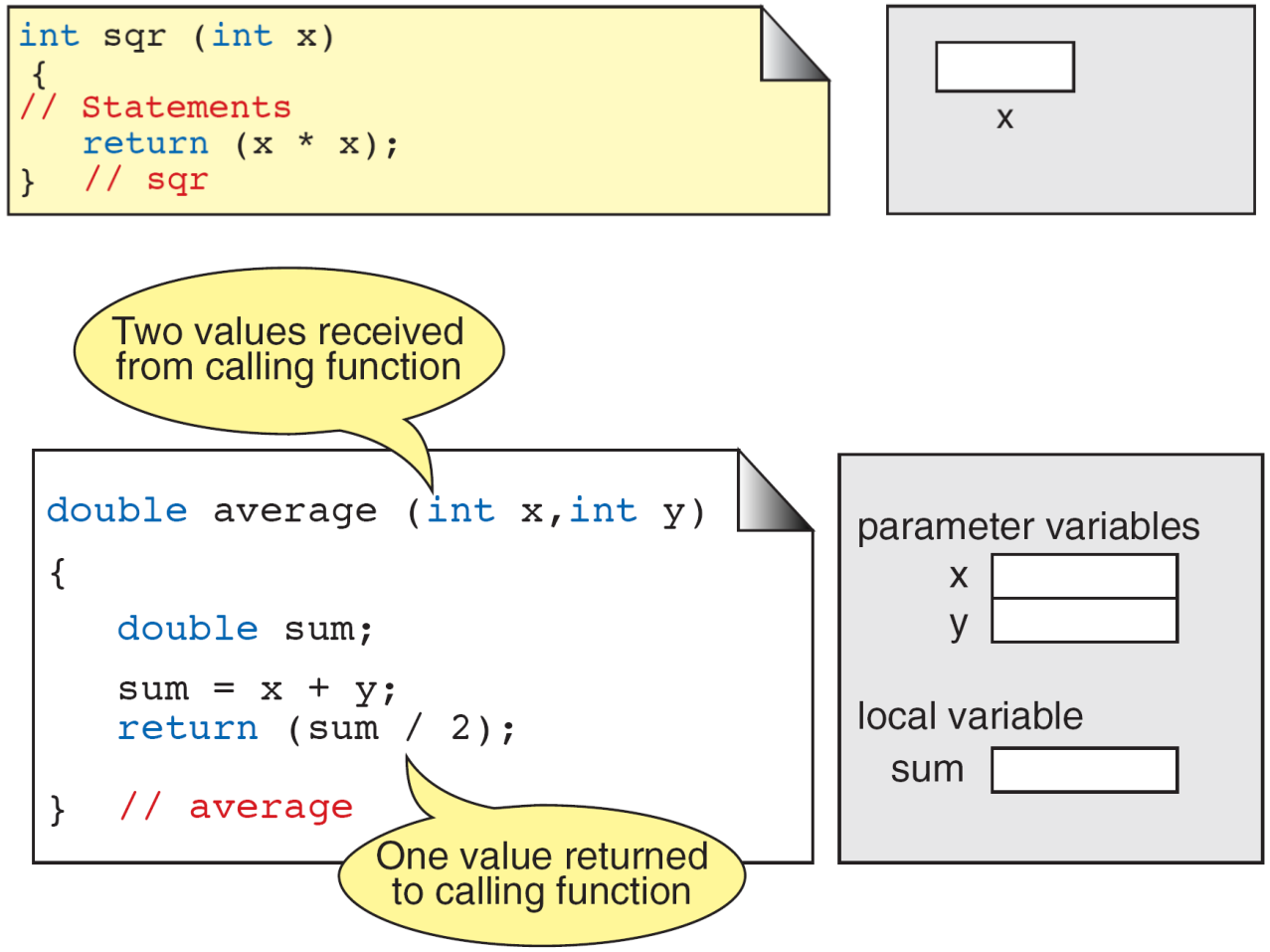

```c
// Function Declaration
int multiply (int multiplier, int multiplicand );

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

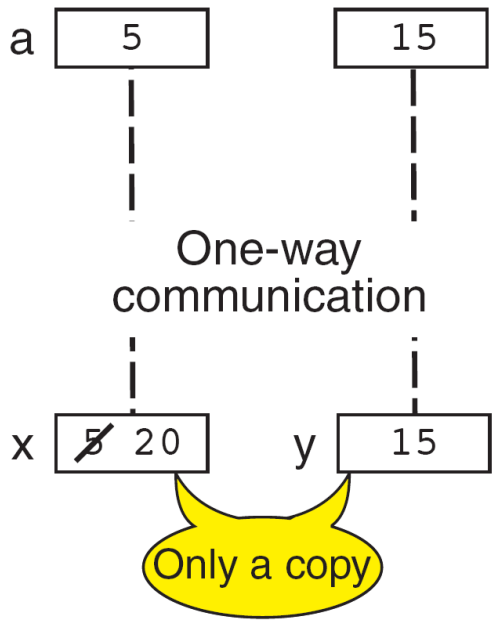


Call-by-Value

```
// 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

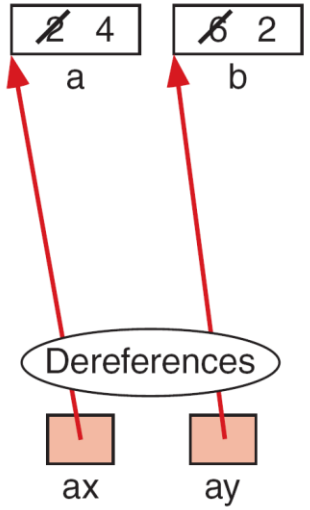
```
   void downFun (int x, int y)
   {
   // Statements
      x = x + y;
      return;
   }  // downFun
```



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

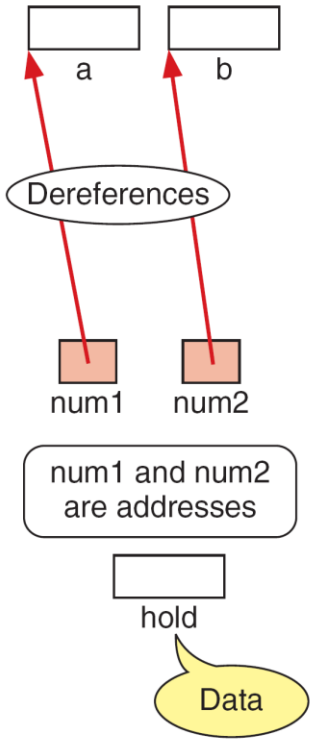
```
// Function Declarations
void exchange (int* num1, int* num2);

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
```

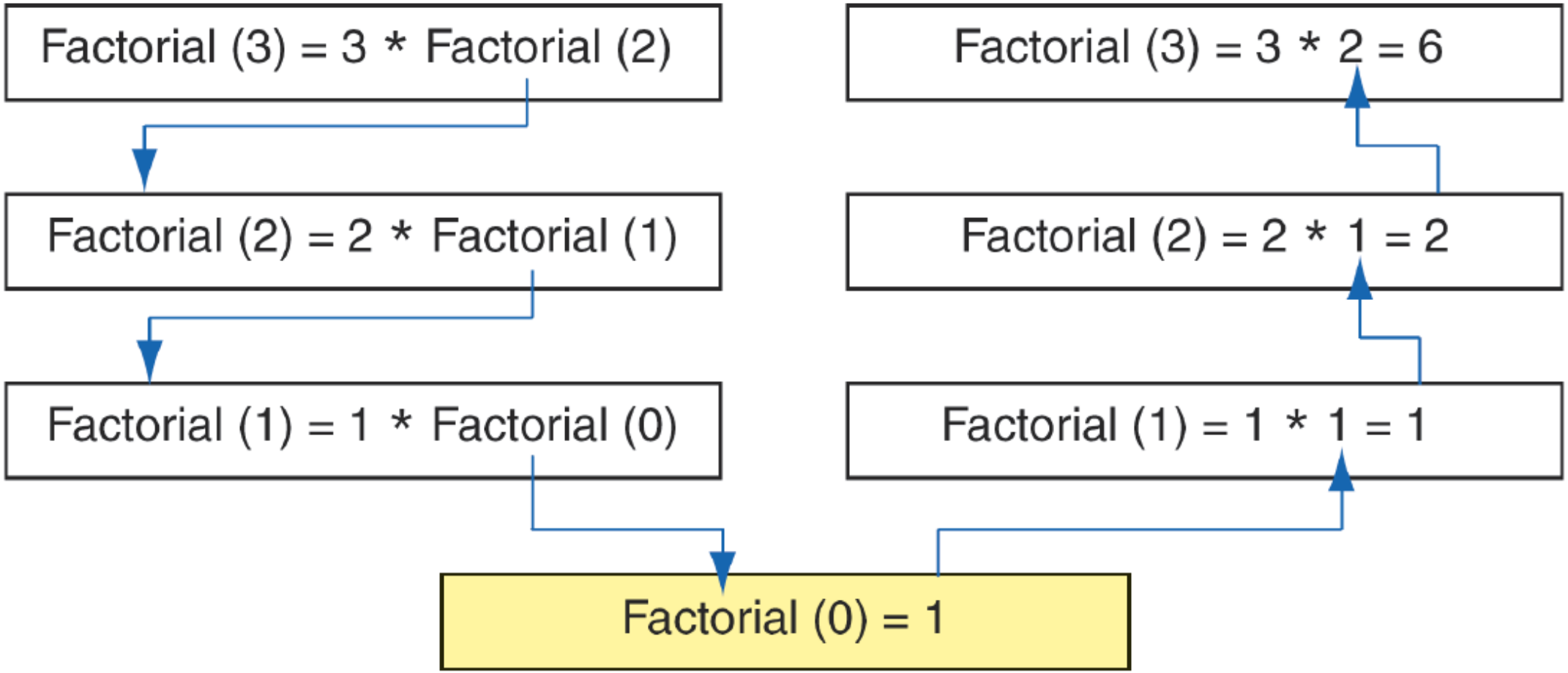
- 중괄호로 싸여져 있는 코드 블럭, 코드 범위
- 한 Scope 내에서 선언된 변수는 그 Scope 범위 안에서만 존재한다.
- 자기보다 상위 Scope의 변수는 볼 수 있지만 하위 Scope의 변수는 볼 수 없다.
- Scope별로 변수를 만들 수 있다.
- 바깥 Scope, 지금 Scope 에 같은 이름의 변수가 있으면 현재 Scope 에 있는 변수를 본다.

오늘 할 것

- 재귀함수(recursive)
- 파일 I/O

Recursion

Factorial Recursively



Iterative vs Recursive

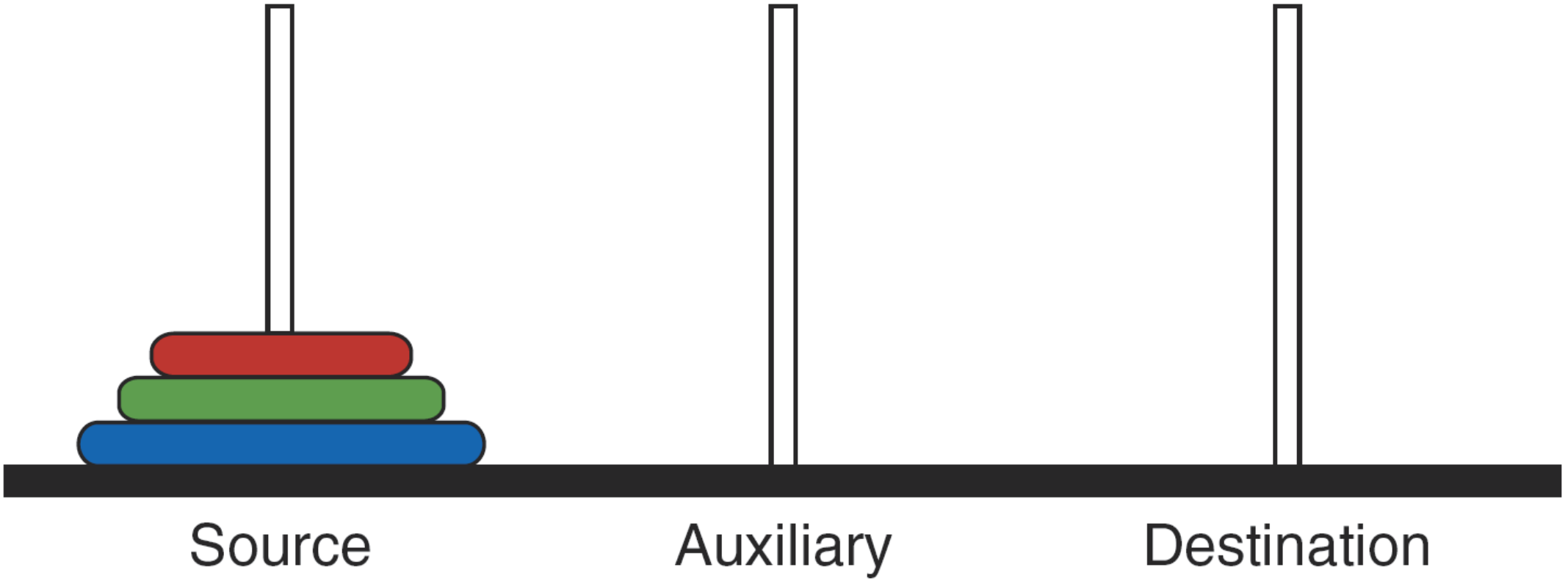
$$\text{factorial}\,(n) = \begin{cases} 1 & \text{if } n = 0 \\ n * \text{factorial}\,(n-1) & \text{if } n > 0 \end{cases}$$

$$\text{factorial}\,(n) = \begin{cases} 1 & \text{if } n = 0 \\ n * (n-1) * (n-2) \ldots 3 * 2 * 1 & \text{if } n > 0 \end{cases}$$

Limitations of Recursion

- Recursive solutions may involve extensive overhead because of function calls
- Each recursive call uses up some of memory allocation
- Possibly duplicate computation, but not always

Tower of Hanoi

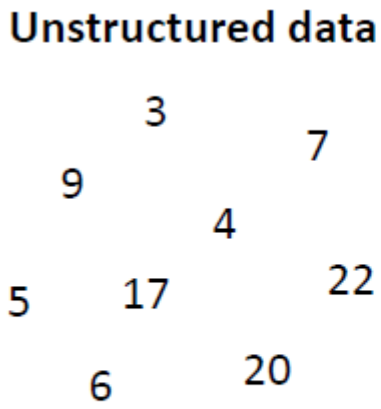


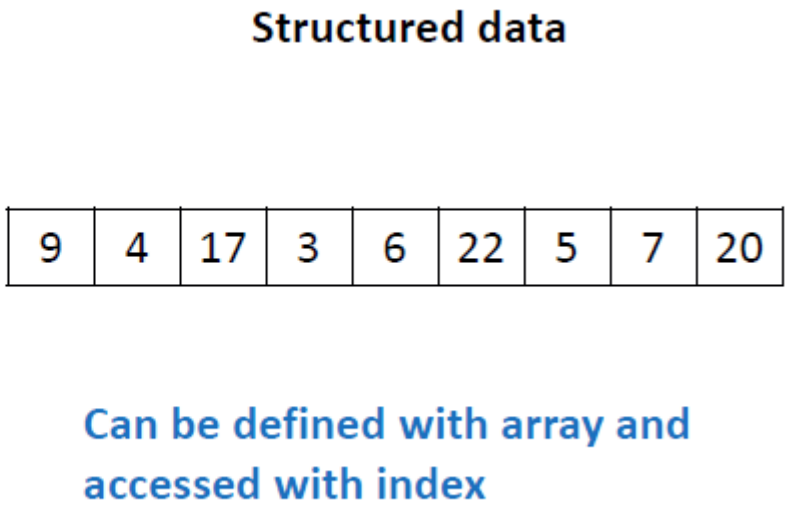
Array 맛보기

자료구조 Data Structure

- A particular way of **storing and organizing data** in a computer so that it can be **used efficiently**.
- 많은 양의 데이터를 효과적으로 관리, 사용하기 위해 (ex. 인터넷 검색)
- 좋은 자료구조는 데이터를 빠르게 검색할 수 있다.

- 종류
  - Matrices
  - Linked lists
  - Priority queues
    - stack, queue, etc.
  - Hash tables
  - Trees and graphs

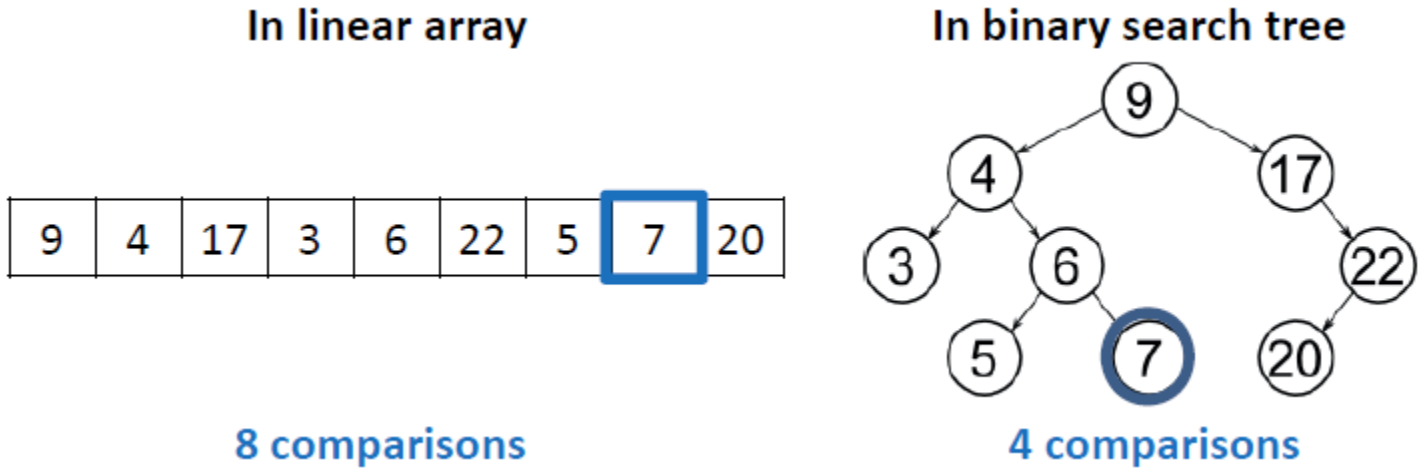




왜 쓰나요

- 많은 데이터를 예쁘게 잘 저장해서
- 쉽고 빠르게 꺼내 쓰려고
- 데이터가 많아지면 찾는데도 오래 걸린다.

- Example: finding a number from a set of numbers
  - How many comparisons do we need to retrieve 7?



Array

Array가 뭔가요

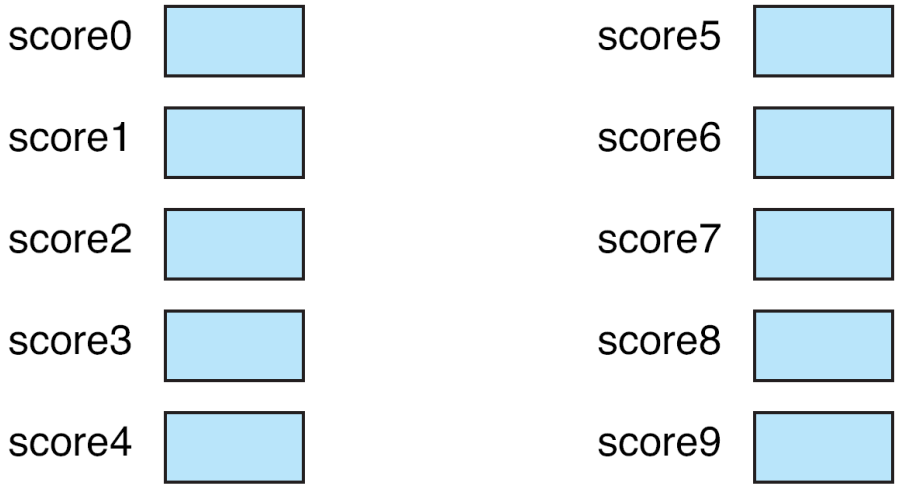
- 연관성 있는 변수들을 하나로 묶어서 보관하는 데이터 타입

Array를 왜 쓰나요?

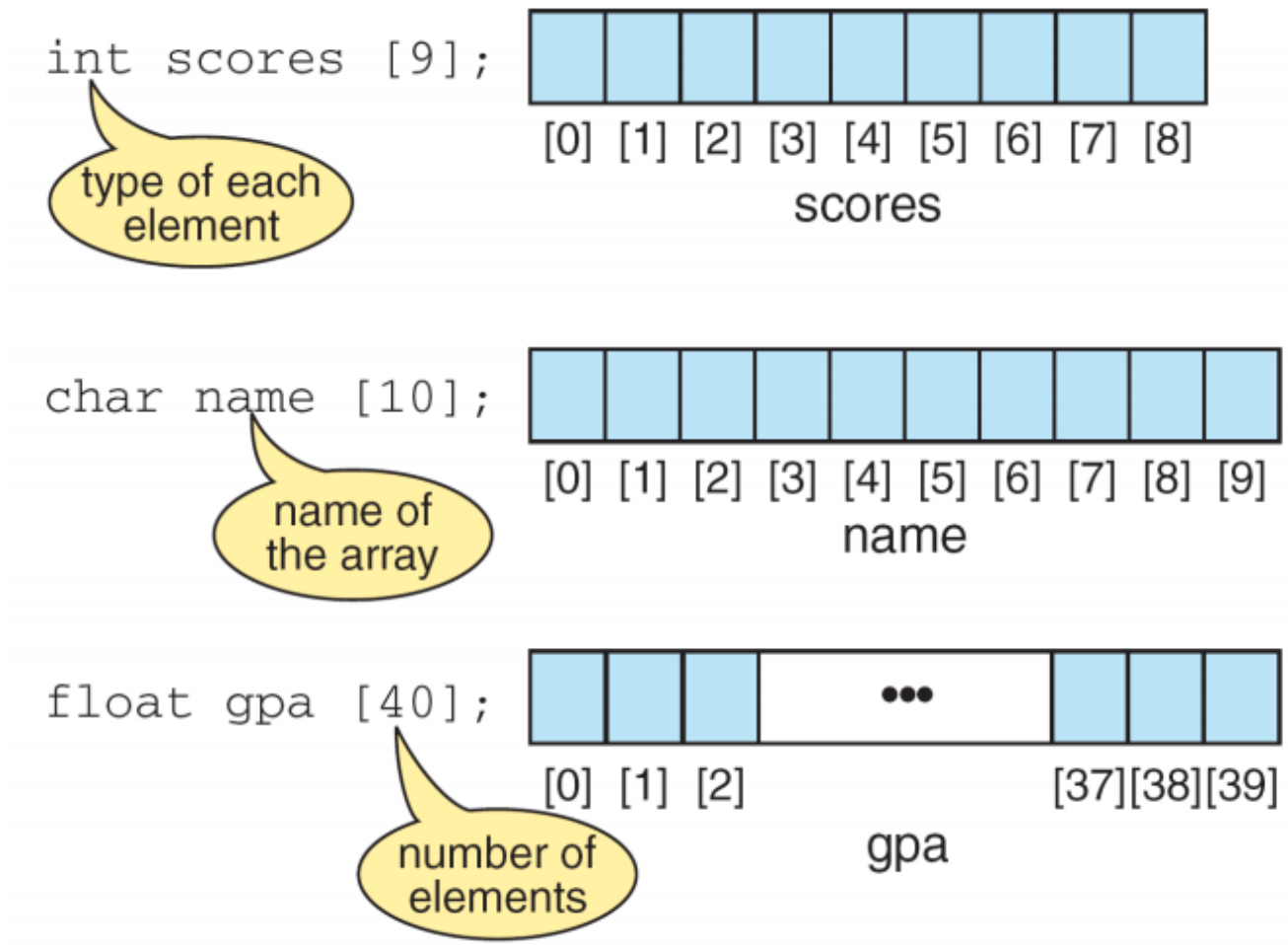
- 여러개의 데이터를 저장해야 할때!
  - 변수를 일일이 만들건가?

- 그렇게 일일이 만들었다고 쳐!
  - 그 여러개에 저장된거를 어떻게 가져올 것인가?

Array!



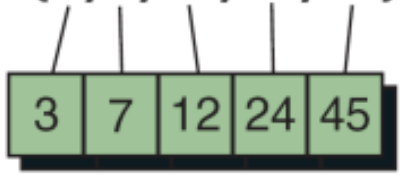
선언


int scores [9];
type of each element
[0] [1] [2] [3] [4] [5] [6] [7] [8]
scores
char name [10];
name of the array
[0] [1] [2] [3] [4] [5] [6] [7] [8] [9]
name
float gpa [40];
number of elements
[0] [1] [2]
[37][38][39]
gpa

초기화

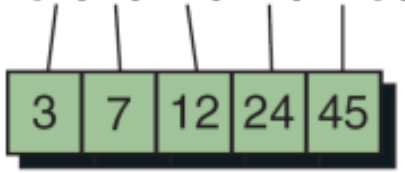
(a) Basic Initialization

```
int numbers[5] = {3,7,12,24,45};
```



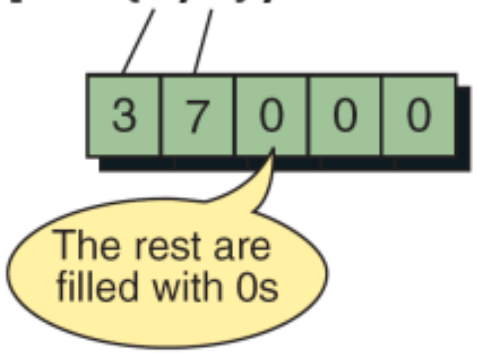
(b) Initialization without Size

```
int numbers[ ] = {3,7,12,24,45};
```



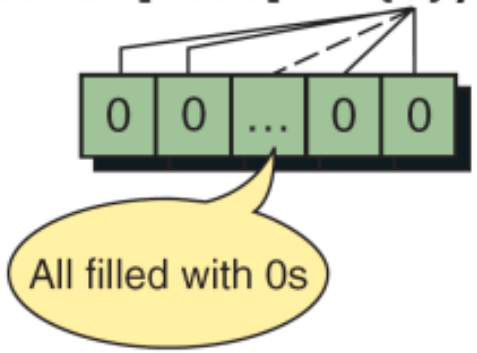
(c) Partial Initialization

```
int numbers[5] = {3,7};
```



(d) Initialization to All Zeros

```
int lotsOfNumbers [1000] = {0};
```



접근

name[index]
배열은 0부터 시작하는 것 주의

```
for (i = 0; i < 9; i++)
    scanf ("%d", &scores[i]);

scores[4] = 23;
printf("%f",gpa[11]);     //배열 gpa의 '12번째' 실수 출력
```

Element address = array address + (sizeof (element) * index)

다음시간

- Array 본론 + Sorting & Searching

- 중간고사 대비 총정리