Week 4

Chapter 3
Control Flow
제어 흐름 = 계산 순서

CSE2018 시스템프로그래밍기초
2016년 2학기

한양대학교 ERICA
컴퓨터공학과 => 소프트웨어학부
도경구

문장
Statement

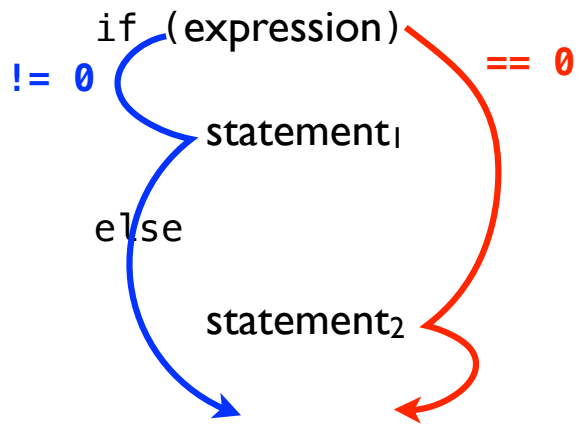
```
x = 0;
i++;
printf(…);
```

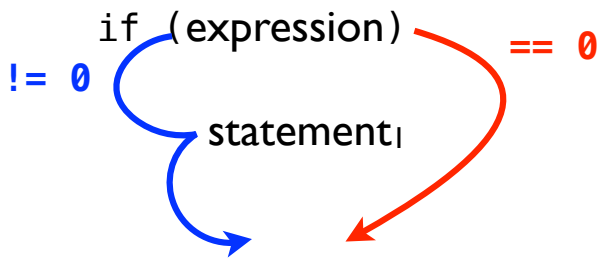
문장의 끝표시
statement terminator

블록
Block

```
{
    declarations
    statements
}
```

If-else

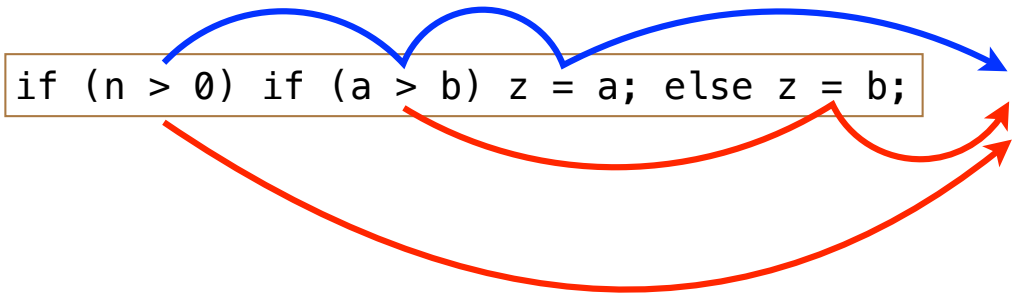


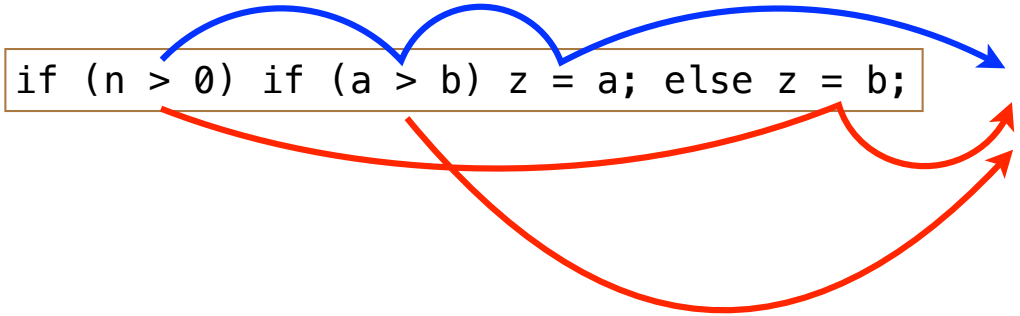



```
if (n > 0)
    if (a > b)
        z = a;
    else
        z = b;
```

모호
Ambiguity

```
if (n > 0)
    if (a > b)
        z = a;
    else
        z = b;
```

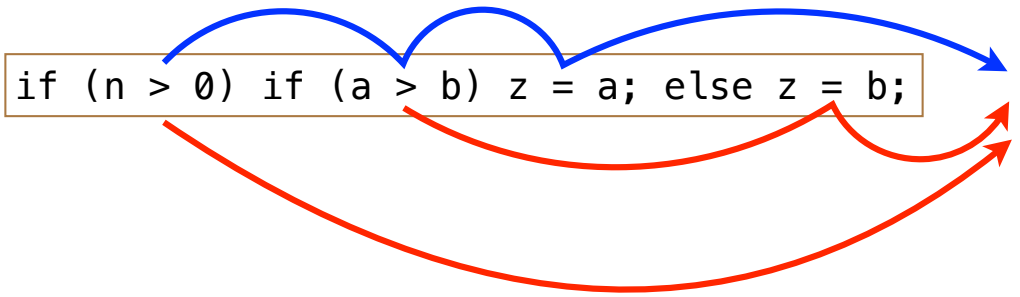




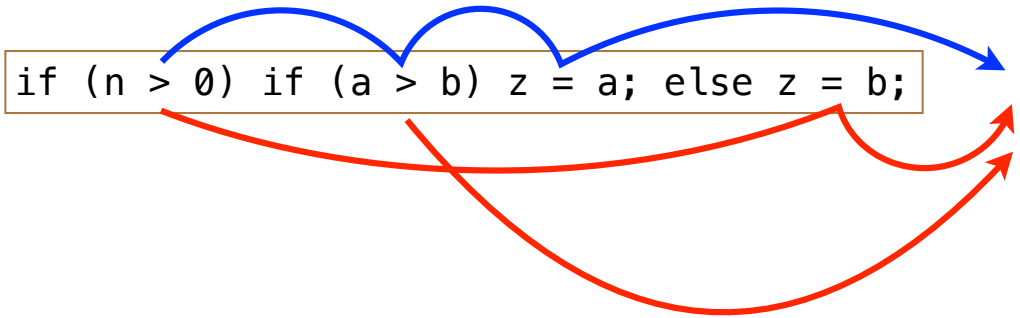
모호
Ambiguity

```
if (n > 0)
    if (a > b)
        z = a;
    else
        z = b;
```

if 가 먼저 걸린 것에서 시작해서 else를 처리함



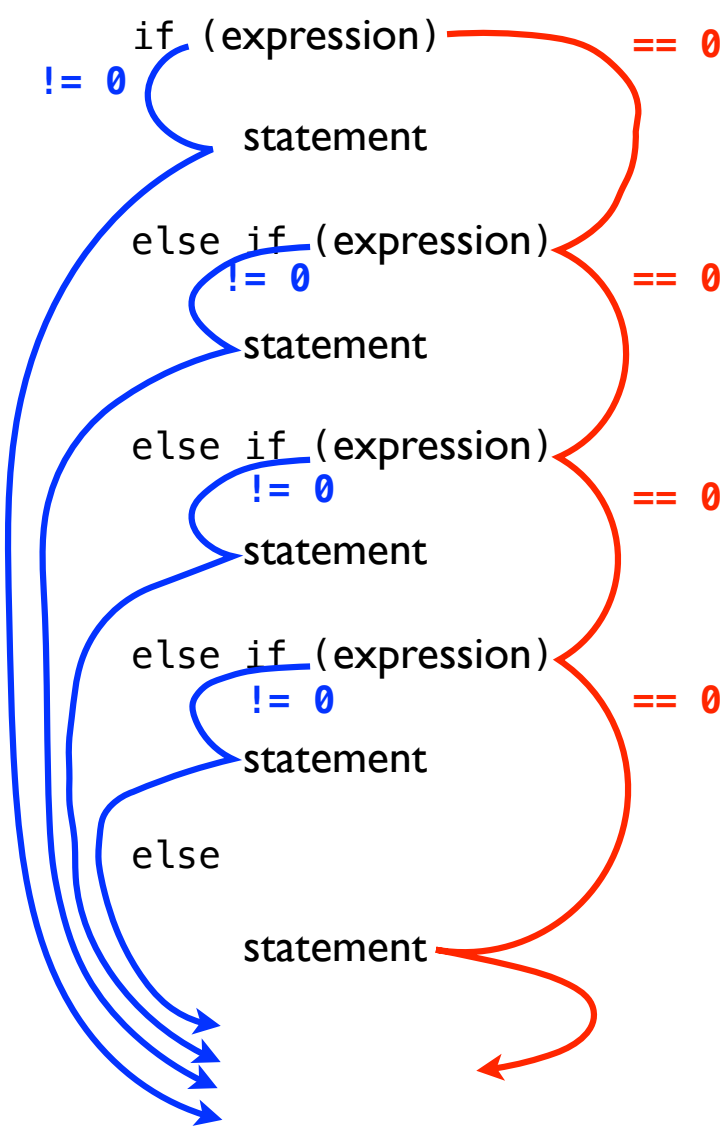

```
if (n > 0) {
    if (a > b)
        z = a;
}
else
    z = b;
```

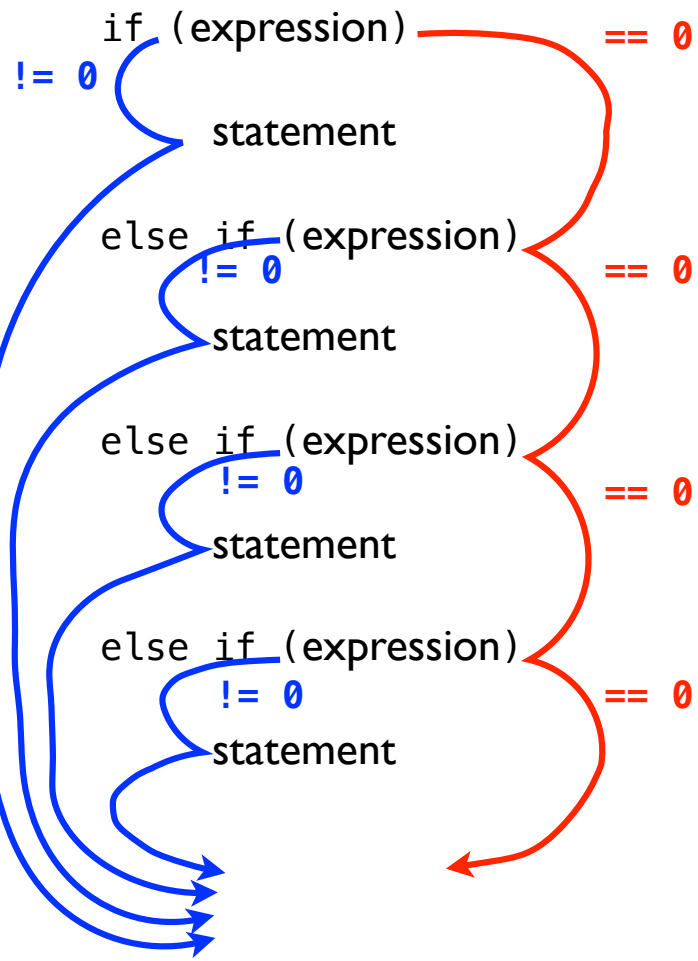


모호
Ambiguity

```
if (n >= 0)
    for (i = 0; i < n; i++)
        if (s[i] > 0) {
            printf("~~~");
            return i;
        }
else
    printf("error – n is negative\n");
```

Else-if


if (expression)
!= 0
== 0
statement
else if (expression)
!= 0
== 0
statement
else if (expression)
!= 0
== 0
statement
else if (expression)
!= 0
== 0
statement
else
statement

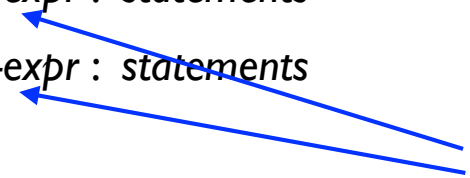

if (expression)
!= 0
== 0
statement
else if (expression)
!= 0
== 0
statement
else if (expression)
!= 0
== 0
statement
else if (expression)
!= 0
== 0
statement

이분검색
Binary Search

```
/* binsearch: find x in v[0] <= v[1] <= ... <= v[n-1] */
int binsearch(int x, int v[], int n) {
    int low, high, mid;

    low = 0;
    high = n - 1;
    while (low <= high) {
        mid = (low + high) / 2;
        if (x < v[mid])
            high = mid - 1;
        else if (x > v[mid])
            low = mid + 1;
        else  /* found match */
            return mid;
    }  /* no match */
    return -1;
}
```

Switch

```
switch (expression) {
    case const-expr : statements
    case const-expr : statements
     . . .
    default : statements
}
```



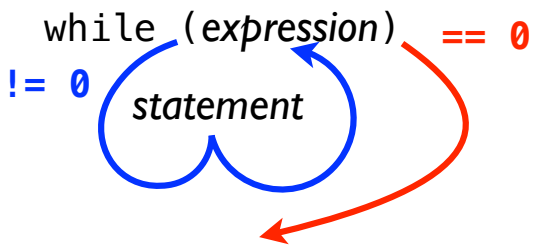
정수값이 되는 식
integer-valued constant expression

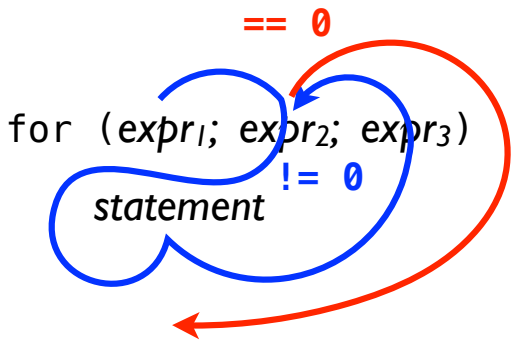
Switch

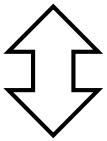
```c
#include <stdio.h>

/* count digits, white space, others */
int main() {
    int c, i, nwhite, nother;
    int ndigit[10];

    nwhite = nother = 0;
    for (i = 0; i < 10; ++i)
        ndigit[i] = 0;
    while ((c = getchar()) != EOF)
        if (c >= '0' && c <= '9')
            ++ndigit[c-'0'];
        else if (c == ' ' || c == '\n' || c == '\t')
            ++nwhite;
        else
            ++nother;

    printf("digits =");
    for (i = 0; i < 10; ++i)
        printf(" %d", ndigit[i]);
    printf("\nwhite space = %d\nother = %d\n",
           nwhite, nother);
}
```

```c
#include <stdio.h>

/* count digits, white space, others */
int main() {
    int c, i, nwhite, nother;
    int ndigit[10];

    nwhite = nother = 0;
    for (i = 0; i < 10; ++i)
        ndigit[i] = 0;
    while ((c = getchar()) != EOF) {
        switch (c) {
            case '0': case '0': case '0': case '0':
            case '0': case '0': case '0': case '0':
            case '0': case '0':
                ndigit[c-'0']++;
                break;
            case '0': case '0': case '0':
                nwhite++;
                break;
            default:
                nother++;
                break;
        }
    }
    printf("digits =");
    for (i = 0; i < 10; i++)
        printf(" %d", ndigit[i]);
    printf("\nwhite space = %d\nother = %d\n",
           nwhite, nother);
}
```

Loop







```
expr1;
while (expr2) {
    statement
    expr3;
}
```

무한반복
infinite loop

```
while (1)
    statement
```

```
for (;;)
    statement
```

Loop
atoi

```
#include <ctype.h>

/* atoi: convert s to integer; version 2 */
int atoi(char s[]) {
    int i, n, sign;

    for (i = 0; isspace(s[i]); i++)
        ;
    sign = (s[i] == '-') ? -1 : 1;
    if (s[i] == '+' || s[i] == '-')
        i++;
    for (n = 0; isdigit(s[i]); i++)
        n = 10 * n + (s[i] - '0');
    return sign * n;
}
```

Loop
Shell sort

교환정렬이랑 비슷함
거리가 멀수록 바뀌는 횟수가 많아짐
교환정렬의 최적화 버전

```
/* shellsort: sort v[0]...v[n-1] into increasing order */
void shellsort(int v[], int n) {
    int gap, i, j, temp;

    for (gap = n/2; gap > 0; gap /= 2)
        for (i = gap; i < n; i++)
            for (j = i - gap; j >= 0 && v[j] > v[j+gap]; j-= gap) {
                temp = v[j];
                v[j] = v[j+gap];
                v[j+gap] = temp;
            }
}
```

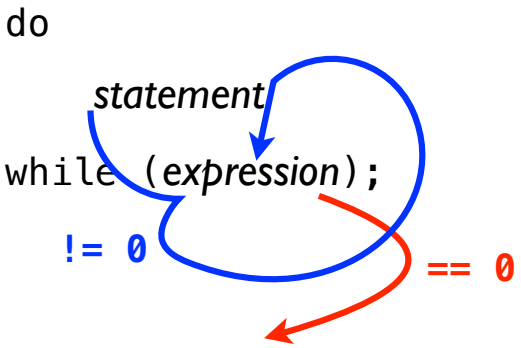
Comma in `for`

`,`

```
#include <string.h>

/* reverse: reverse string s in place */
void reverse(char s[]) {
    int c, i, j;

    for (i = 0, j = strlen(s)-1; i < j; i++, j--) {
        c = s[i];
        s[i] = s[j];
        s[j] = c;
    }
}
```

```
#include <string.h>

/* reverse: reverse string s in place */
void reverse(char s[]) {
    int c, i, j;

    for (i = 0, j = strlen(s)-1; i < j; i++, j--) {
        c = s[i], s[i] = s[j], s[j] = c;
    }
}
```

Loop




```
/* itoa: convert n to characters in s */
void itoa(int n, char s[]) {
    int i, sign;

    if ((sign = n) < 0)  /* record sign */
        n = -n;          /* make n positive */
    i = 0;
    do {  /* generate digits in reverses order */
        s[i++] = n % 10 + '0';  /* get next digit */
    } while ((n /= 10) > 0);    /* delete it */
    if (sign < 0)
        s[i++] = '-';
    s[i] = '\0';
    reverse(s);
}
```

break

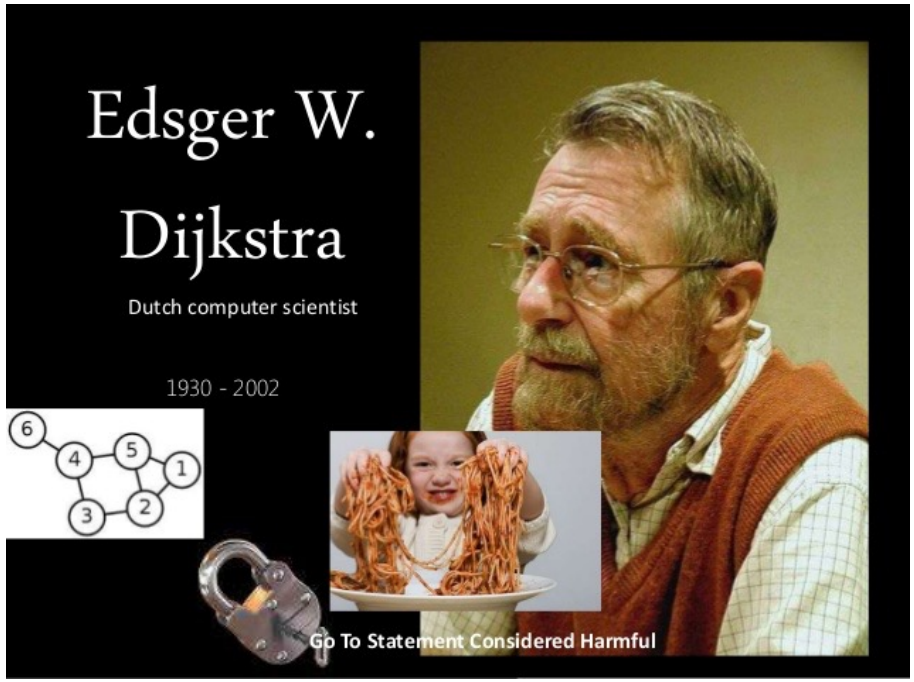
```
/* trim: remove trailing blanks, tabs, newlines */
int trim(char s[]) {
    int n;

    for (n = strlen(s)-1; n >= 0; n--)
        if (s[n] != ' ' && s[n] != '\t' && s[n] != '\n')
            break;
    s[n+1] = '\0';
    return n;
}
```

continue

```
for (i = 0; i < n; i++) {
    if (a[i] < 0)  /* skip negative elements */
        continue;
    . . .  /* do positive elements */
}
```

goto **label**

```
    for ( . . . )
        for ( . . . ) {
            . . .
            if (disaster)
                goto error;
        }
    . . .

error:
    clean up the mess
```



Go To Statement Considered Harmful

Key Words and Phrases: go to statement, jump instruction, branch instruction, conditional clause, alternative clause, repetitive clause, program intelligibility, program sequencing
CR Categories: 4.22, 5.23, 5.24

EDITOR:

For a number of years I have been familiar with the observation that the quality of programmers is a decreasing function of the density of **go to** statements in the programs they produce. More recently I discovered why the use of the **go to** statement has such disastrous effects, and I became convinced that the **go to** statement should be abolished from all "higher level" programming languages (i.e. everything except, perhaps, plain machine code). At that time I did not attach too much importance to this discovery; I now submit my considerations for publication because in very recent discussions in which the subject turned up, I have been urged to do so.

My first remark is that, although the programmer's activity ends when he has constructed a correct program, the process taking place under control of his program is the true subject matter of his activity, for it is this process that has to accomplish the desired effect; it is this process that in its dynamic behavior has to satisfy the desired specifications. Yet, once the program has been made, the "making" of the corresponding process is delegated to the machine.

My second remark is that our intellectual powers are rather geared to master static relations and that our powers to visualize processes evolving in time are relatively poorly developed. For that reason we should do (as wise programmers aware of our limitations) our utmost to shorten the conceptual gap between the static program and the dynamic process, to make the correspondence between the program (spread out in text space) and the process (spread out in time) as trivial as possible.

Let us now consider how we can characterize the progress of a process. (You may think about this question in a very concrete manner: suppose that a process, considered as a time succession of actions, is stopped after an arbitrary action, what data do we have to fix in order that we can redo the process until the very same point?) If the program text is a pure concatenation of, say, assignment statements (for the purpose of this discussion regarded as the descriptions of single actions) it is sufficient to point in the

M. Broy et al. (eds.), *Pioneers and Their Contributions to Software Engineering*

goto **label**

```
    for (i = 0; i < n; i++)
        for (j = 0; j < m; j++)
            if (a[i] == b[j])
                goto found;
        /* did not find any common element */
    . . .

found:
    /* got one: a[i] == b[j] */
    . . .
```

```
    found = 0;
    for (i = 0; i < n && !found; i++)
        for (j = 0; j < m && !found; j++)
            if (a[i] == b[j])
                found = 1;
    if (found)
        /* got one: a[i-1] == b[j-1] */
        . . .
    else
        /* did not find any common element */
        . . .
```