Control Flow

백 윤 철

Contents

Statements and Blocks

- Semicolon is statement terminator

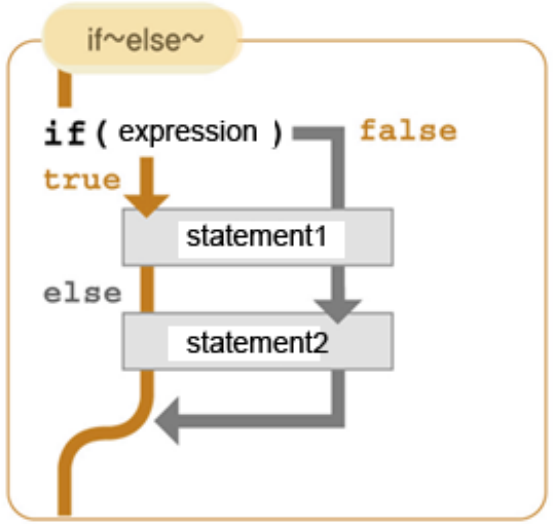
```
x = 0;
i++;
printf(...);
```

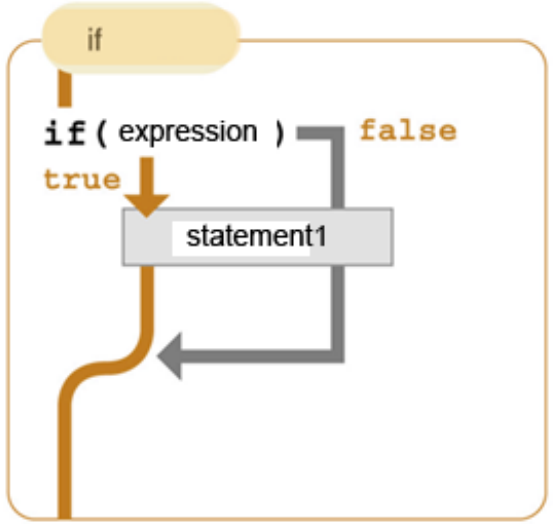
- Braces { and } are used to group declarations and statements together into a compound statement, or block.
- There is no semicolon after the right brace that ends a block.

If-else

- if-else statement is used to express decision.

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





If-else

- Ambiguity of else

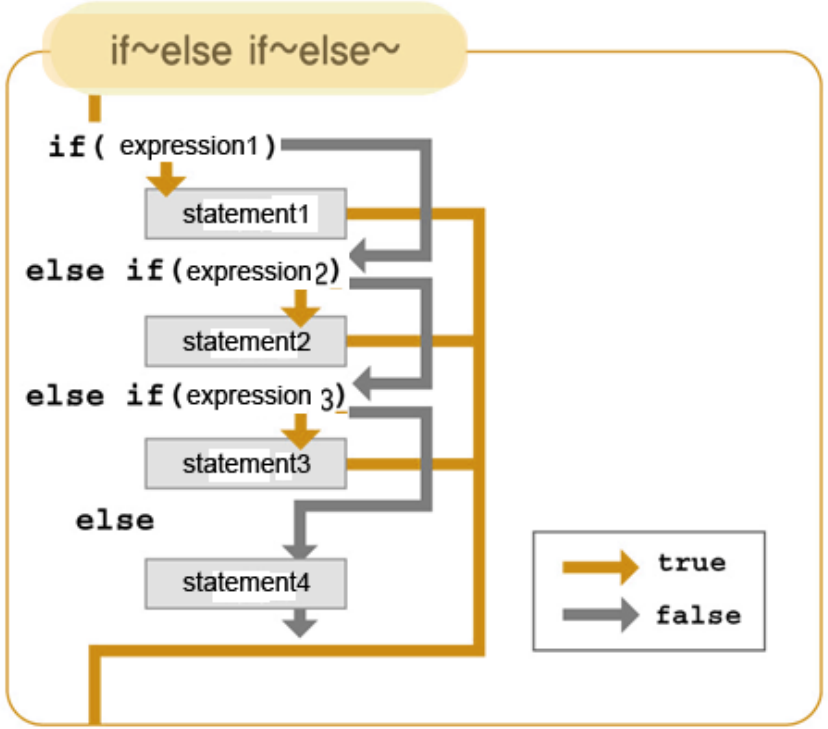
```
if (n > 0)
    if (a > b)
        z = a;
    else
        z = b;
```

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

Else-if

- construction

```
if (expression)
    statement
else if (expression)
    statement
else if (expression)
    statement
else if (expression)
    statement
else
    statement
```



Else-if

- Binary search 이진 탐색

```
/* 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;
    }
    return -1;  /* no match */
}
```

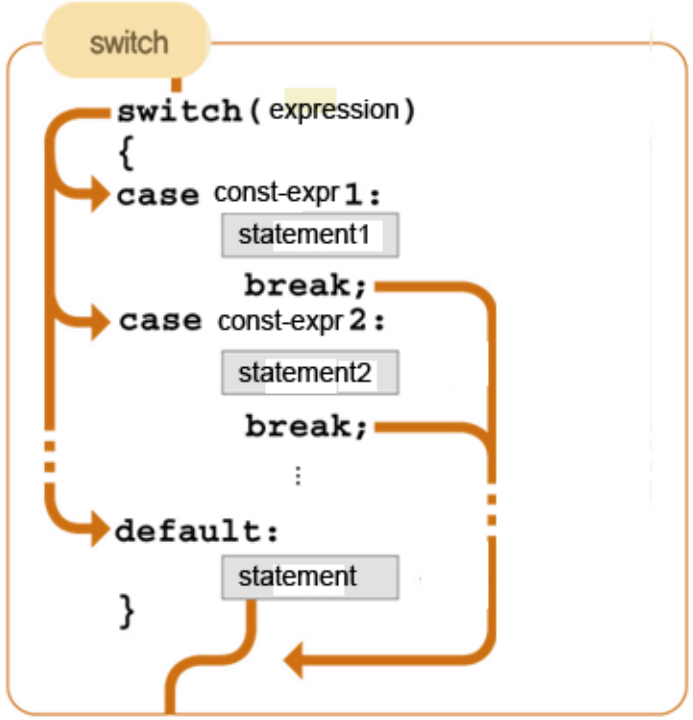
$\frac{1}{2}$

에러반환

Switch

- Multiway decision

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



- Default is optional
- Break causes an exit

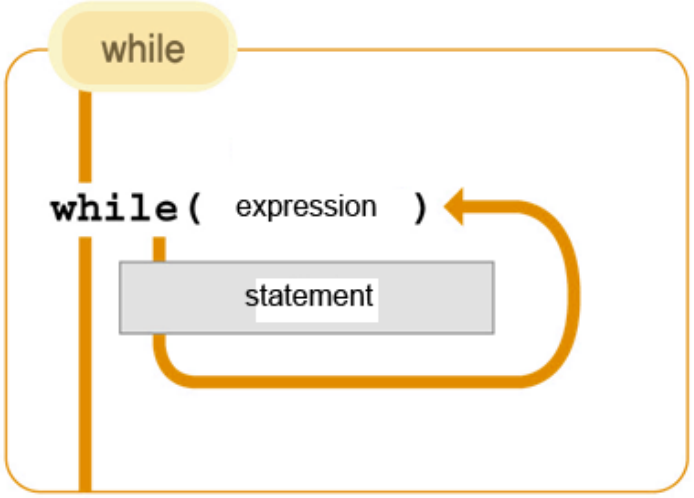
Switch

```
#include <stdio.h>

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

    nwhite = nother = 0;
    for (i = 0; i < 10; i++)
        ndigit[i] = 0;
    while ((c = getchar()) != EOF) {
        switch (c) {
        case '0': case '1': case '2': case '3': case '4':
        case '5': case '6': case '7': case '8': case '9':
            ndigit[c-'0']++;
            break;
        case ' ':
        case '\n':
        case '\t':
            nwhite++;
            break;
        default:
            nother++;
            break;
        }
    }
    printf("digits =");
    for (i = 0; i < 10; i++)
        printf(" %d", ndigit[i]);
    printf(", white space = %d, other = %d\n",
        nwhite, nother);
    return 0;
}
```

Loops-While and For

- The expression is evaluated, if it is non-zero, statement is evaluated and expression is re-evaluated.
- The cycle continues until expression becomes zero

```
while (expression)
    statement
```

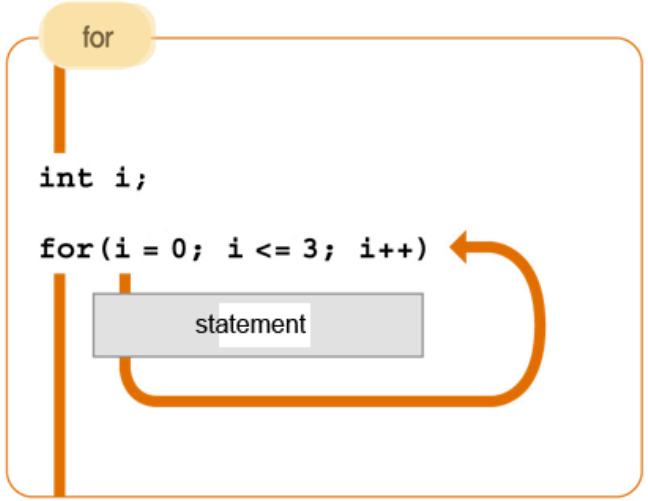


Loops-While and For

- for

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

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



Loops-While and For

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

공백 = 0이상인수 공백아님=0 반환

or

Loops-While and For

- shell sort 셸 정렬

```
/* 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;
            }
}
```

Loops-While and For

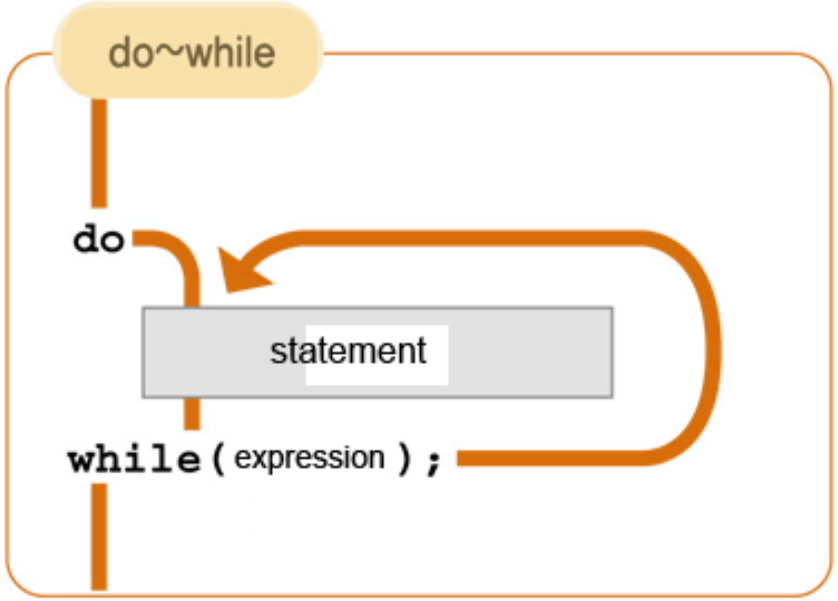
- reverse()

```
#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;
    }
}
```

Loops-Do-while

- The body is always executed at least once.

```
do
    statement
while (expression);
```



Loops-Do-while

- itoa()

sign	-123
n	0

s	‘3’	‘2’	‘1’	‘-’	\0

```
/* 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 reverse 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 and Continue

- The break statement provides an early exit from for, while, and do, just as from switch.
- A break causes the innermost enclosing loop or switch to be exited immediately.

```
/* 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;
}
```

H	e	l	l	o	``	\t	\n	\0

Break and Continue

- Continue causes the next iteration of the enclosing for, while, or do loop to begin.
- In the while and do, this means that the test part is executed immediately.

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

Goto and Labels

- breaking out of two or more loops at once
- in practice it is almost always easy to write code without it

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

error:
    clean up the mess
```

Goto and Labels

- A label has the same form as a variable name, and is followed by a colon.
- It can be attached to any statement in the same function as the goto.
- The scope of a label is the entire function.

Goto and Labels

- the problem of determining whether two arrays a and b have an element in common.

```
    for (i = 0; i < n; i++)
        for (j = 0; j < m; j++)
            if (a[i] == b[j])
                goto found;
    /* didn't find any common element */
    ...
found:
    /* got one:  a[i] == b[j] */
    ...
```

Goto and Labels

- Code involving a goto can always be written without one
- though perhaps at the price of some repeated tests or an extra variable.

```
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
    /* didn't find any common element */
    ...
```

- code that relies on goto statements is generally harder to understand and to maintain than code without gotos

정리

- Statements and Blocks
- If-Else
- Else-If
- Switch
- Loops-While and For
- Loops-Do-While
- Break and Continue
- Goto and Labels

끝