[DASF004-42]

Basis and Practice in Programming

(프로그래밍 기초와 실습)

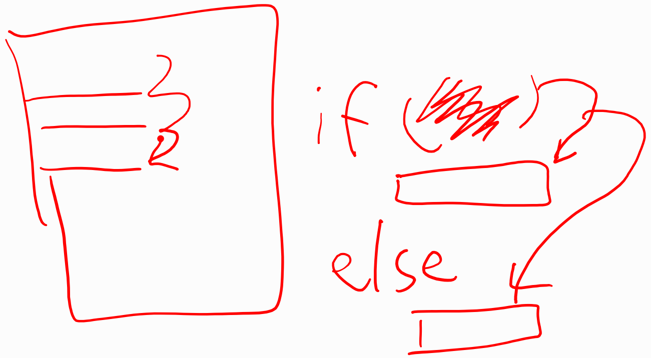
Spring 2022

HYUNGJOON KOO

SUNG KYUN KWAN UNIVERSITY(SKKU)

Program Control (Recap)

- **Program control**
  - Specifying the order of statements for program execution
- **Sequential execution** 순차
  - Line-by-line execution
  - Order of statements in a code is **equal** to the execution order.
- **Transfer of program control**
  - Execution order is **different** from what you can see in the code.
- **Control flow** 제어흐름 CFG
  - The order in which individual statements are executed or evaluated.



Control Structures (Recap)

Three types of control structures

✓ - Sequence structure

✓ - Selection structure

  » if, if…else, switch/case

✓ - Iteration structure 반복구문

  » while, do…while, for

- C has only seven control statements.

Iteration Statement

Iteration statement

- makes an action to be repeated when a certain condition is true.
- **Loop condition**: A loop is a group of instructions that the computer executes repeatedly while its *loop condition* remains true.

Three means of repetition (iteration)

- Definite repetition: Exact number of iterations is known. 정의된
- Indefinite repetition: Number of iterations is determined on runtime. 부정의
- Infinite repetition: The number of iteration is infinite. 무한
  - » Intentionally? A bug?

Three statements for iterations

- `while, do…while, for`

`do-while` **statement**

`while / do-while` Statement - Systax

SUNG KYUN KWAN UNIVERSITY(SKKU)

Syntax

- while

□ Loop Conditions

참

```
while(logical_expression_1) {
    /* When logical_expression_1 is true */
    (statements to be repeated)
}
```

- do-while

```
do {
/* Loop exits when logical_expression_1 is false */
    (statements to be repeated)
} while(logical_expression_1)
```

while Statement

Execution order of while statement

- The loop-continuation condition is tested at the beginning of the loop before the body of the loop is performed.

loop condition

```
while(logical_expression_1) {
    /* When logical_expression_1 is true */
    (statements to be repeated)
}
```

body

- logical_expression_1 is evaluated first.
- If it is true, the body of the while statement is executed.
- After each iteration, logical_expression_1 is checked for true. The body is repeatedly executed under the true condition of logical_expression_1.
- When it becomes false, the program control escapes the loop and the next statement in order is executed.

while Iteration example

- **Example: finding the first power of 3 larger than 100**

```
product = 3;

while ( product <= 100 ) {

   product = 3 * product;

} // end while
```

3, 9, 27, 81, 243

243 [illegible]

- The loop starts with the product variable as 3.
- The loop condition is true for that value and the body of while is executed.
- The value of product increases as 9, 27 and 81.
- When product becomes 243, the loop condition becomes false, which terminates the iteration. (Check that this value is what we are looking for.)
- Program execution continues with the next statement after the while.

while example

- **Easiest example with while**
  - Variations with different loop conditions and strides

```c
#include <stdio.h>

int main() {
    int num = 1;
    while (num <= 10) {
        printf("%d\n", num);
        num = num + 1;
    }
    return 0;
}
```

2

↳ num ++;

num += 1;

```
1
2
3
4
5
6
7
8
9
10
```

while example

- **Example: finding the first power of 3 larger than 100**

```
#include <stdio.h>

int main() {

  int product = 3;

  while ( product <= 100 ) {

    product = 3 * product;

  } // end while


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

  return 0;

}
```

~~3~~ ~~9~~ ~~27~~ ~~81~~

243

```
243
```

do…while Iteration

Execution order of do…while statement

- The do...while statement tests the loop-continuation condition after the loop body is performed. (Very similar to the execution order of while statement, so not written here.)

```
do {
/* Loop exits when logical_expression_1 is false */
     (statements to be repeated)
} while(logical_expression_1);
```

Difference against the while statement

- while: the loop condition is tested first. The body might never be executed.
- do...while: the loop condition is checked after the body is performed.
  The body has at least one chance to be executed.

do…while **Iteration**

Transition from while to do…while

- In most cases, converting while statements to do…while statements can be done easily.
- However, you need to be very cautious of the first execution of the body.

```
product = 3;
while ( product <= 100 ) {
   product = 3 * product;
} // end while
```

✱ With while

product = 3

With do…while

200

```
product = 3;
do {
   product = 3 * product;
} while( product <= 100 ) // end do…while
```

600

do…while Iteration example

Example: finding the first power of 3 larger than 100

```
product = 3;
do {
   product = 3 * product;
} while( product <= 100 ); // end do…while
```

- The loop starts with the `product` variable as 3 and the body is executed.
- The value of product increases as 9, 27 and 81.
- When the loop condition becomes false, which terminates the iteration.

- **CHECK: What happens when the initial value of `product` is 243 instead of 3?**

do…while **example**

```
#include <stdio.h>


int main()
{
    int num = 1;
    do {
        printf("%d\n", num);
        num = num + 1;
    } while (num <= 10);
    return 0;
}
```

```
1
2
3
4
5
6
7
8
9
10
```

do…while **example**

```
#include <stdio.h>


int main() {
    int product = 3;
    do {
        product = 3 * product;
    } while( product <= 100 ); // end do…while
    printf("%d\n", product);
    return 0;
}
```

```
243
```

Types of iteration: Definite

- **Definite repetition**
  - Exact number of iterations is known.

    Compilation time
  - Counter-controlled iteration
    - Counter: Specifying the number of times for a set of statements that should be executed

```
int counter = 1;
while ( counter <= 10 ) {
     printf("%d\n", counter);
     counter++;
}
```

Types of iteration: Indefinite

Indefinite repetition

- Number of iterations is determined on runtime.
- Sentinel-Controlled Iteration
  - Sentinel value: Indicating "end of data entry"
  - Other names: a signal value, a dummy value, a flag value
  - The sentinel value must be chosen so that it cannot be confused with an acceptable input value.

```
int num;
scanf("%d", &num);
while ( num != -1 ) {        // sentinel value is -1
    printf("%d\n", num);
    scanf("%d", &num);
}
```

Types of iteration: Infinite

Infinite repetition

- The number of iteration is infinite.
- When we use it in our program, we normally terminate the loop using ***break*** statement under some specific conditions.

```
// An erroneous example
while ( 1 ) {
    printf("while (1)");
}
```

true

if( ) {

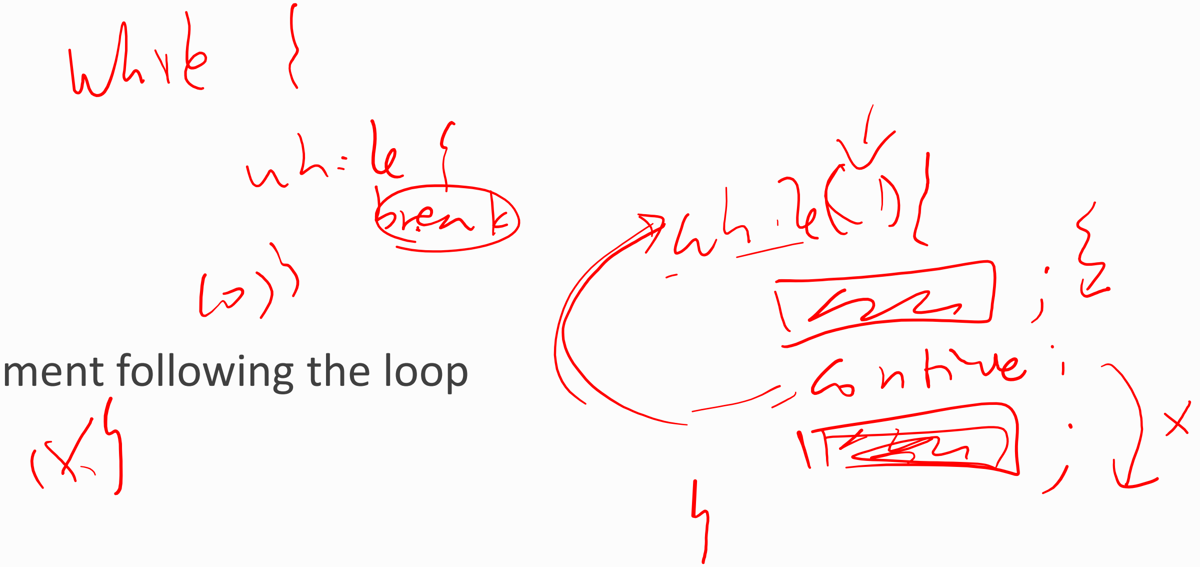
break;

}

break / continue

- `break` **and** `continue` **are used for managing program control explicitly.**

- `break` **breaks loops.**
  - `break` has two usages.
  - With loops: Exiting from innermost one
    - » The program control resumes at the next statement following the loop
  - With switch: Termination of a case



- `continue` **continues loops.**
  - With `continue`, loop continues next iteration ignoring following code in the code block.
  - In `while` and do…`while` statements, the loop-continuation test is evaluated immediately after the continue statement is executed.

break / continue example

```
#include <stdio.h>
int main()
{
    int num = 0, sum = 0;
    while (num < 10) {
        sum += num++;
        printf("before if\n");
        if (sum < 20) {
            printf("num:%d sum:%d (continue)\n", num, sum);
            continue;
        } else {
            printf("num:%d sum:%d (break)\n", num, sum);
            break;
        }
        printf("after else\n");
    }
}
```

sum = num + sum;

5 = 2 + 1

(unreachable code)

CHECK1: What is the first value added to sum?
CHECK2: Is the loop escaped by the loop-continuation condition?
CHECK3: Find a line never being executed.

Indefinite iteration example

```
#include <stdio.h>


int main()
{
    int num = 1;
    while (num != 10) {
        printf("Input a number(for exit, type 10): ");
        scanf("%d", &num);
        printf("You type %d.\n", num);
    }
    return 0;
}
```

Exercise – Loop with increment op.

- Fill in the blank to print the result on the right side.

```
#include <stdio.h>

int main() {

    int num = 1;

    while (num <= 10) {

        printf("%d\n", [BLANK] );

    }

    return 0;

}
```

```
1
2
3
4
5
6
7
8
9
10
```

Follow-up question on the exercise

- **We have several candidate positions for '++' increment operator. (On 5th and 6th line, before and after num variable.) What will be the results for each position?**

```
#include <stdio.h>
int main() {
    int num = 1;
    while (num <= 10) {
        printf("%d\n", num++);
    }
    return 0;
}
```

++num.

~~1~~
2
3

Reading Materials

Basic concept for the while iteration statement

- An iteration statement allows you to specify that an action is to be repeated while some condition remains true. The pseudocode statement describes the iteration that occurs during a shopping trip. The condition, "there are more items on my shopping list" may be true or false. If it’s true, then the action, "Purchase next item and cross it off my list" is performed. This action will be performed repeatedly while the condition remains true.

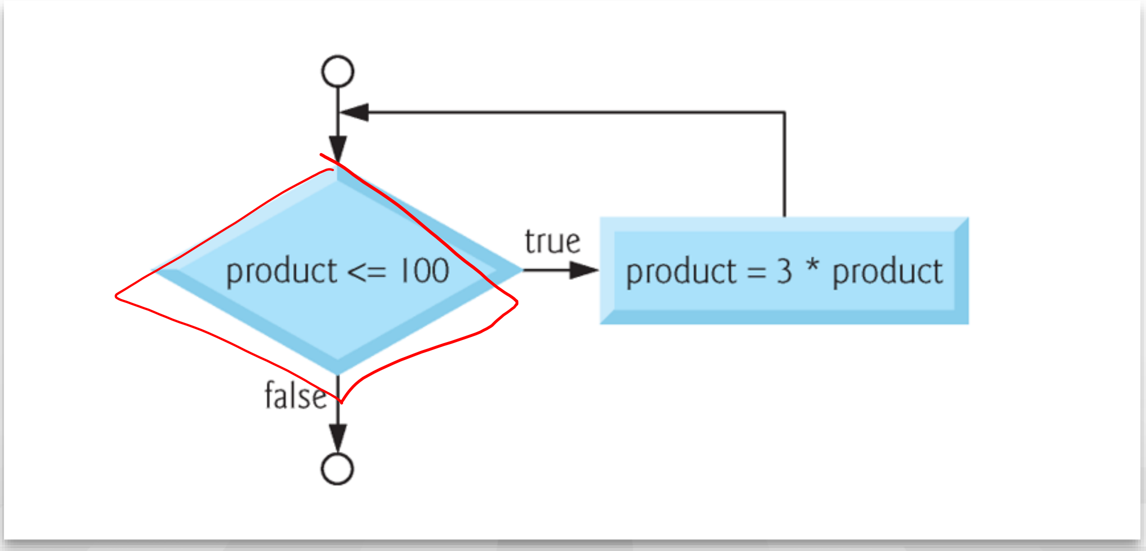
```
While there are more items on my shopping list
   Purchase next item and cross it off my list
```

- Eventually, the condition will become false (when the last item on the shopping list has been purchased and crossed off the list). At this point, the iteration terminates, and the first pseudocode statement after the iteration structure is executed.

Reading Materials

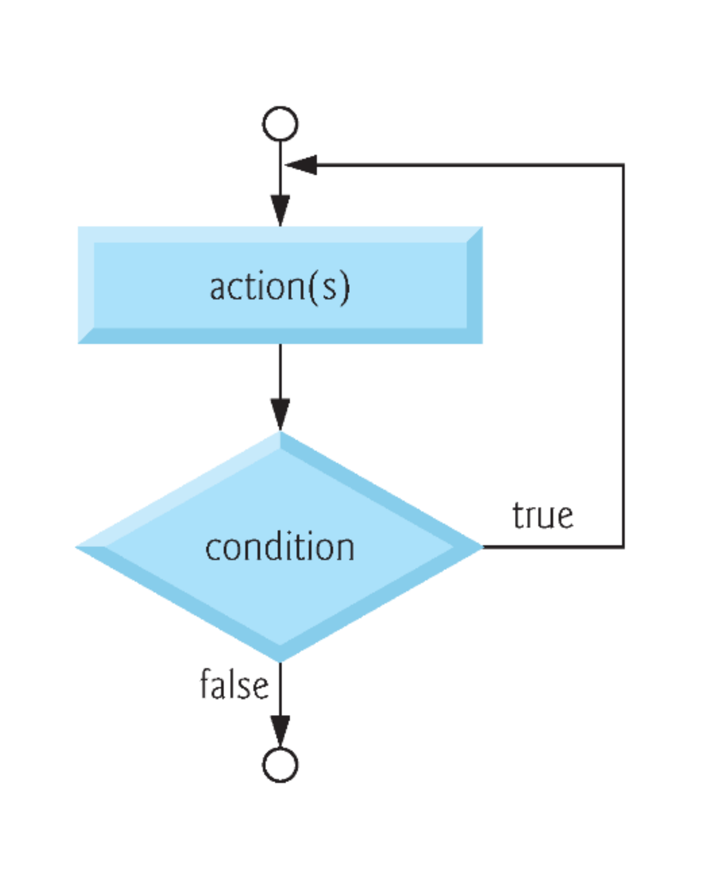
- **Example code and corresponding flowchart**

```
product = 3;
while ( product <= 100 ) {
   product = 3 * product;
} // end while
```



do...while Iteration Statement

- The do...while iteration statement is similar to the while statement. In the while statement, the loop-continuation condition is tested at the beginning of the loop before the body of the loop is performed. The do...while statement tests the loop-continuation condition after the loop body is performed. Therefore, the loop body will be executed at least once. When a do...while terminates, execution continues with the statement after the while clause.



Hands-on Lab

- while.c
- do_while.c
- while_with_scanf.c
- break_continue.c

[DASF004-42]

Basis and Practice in Programming

(프로그래밍 기초와 실습)

Spring 2022

HYUNGJOON KOO

SUNG KYUN KWAN UNIVERSITY(SKKU)

for **statement**

Counter-controlled Iteration

Counter-controlled iteration

- Definite iteration (or definite repetition)
  - vs. indefinite iteration (sentinel-controlled iteration)
- The exact number of loop execution is known in advance.

Counter-controlled iteration requires:

- The name of a control variable (or loop counter).
- The initial value of the control variable.
- The increment (or decrement) by which the control variable is modified each time through the loop.
- The condition that tests for the final value of the control variable (i.e., whether looping should continue).

for Statement

- The for iteration statement handles all the details of counter-controlled iteration.
- Syntax

i < 10

i = 0 증감 i++

```
for(initialization; condition; increment) {
        /* When condition is true */
        (code block)
}
```

- initialization
  - executed first and once
  - a name of a control variable and its initial value
    - Multiple control variables are acceptable.
- Condition
  - loop condition

for Statement

- **The `for` iteration statement handles all the details of counter-controlled iteration.**
- **Syntax**

```
for(initialization; condition; increment) {
        /* When condition is true */
        (code block)
}
```

for (; ;) ⟺ while(1)

- Increment
  - update any loop control variables
  - Control variables are modified each time through the loop
- Execution order:
  - initialization → condition → code block → increment → condition → code block → increment → condition → code block → …

Other cases of `for` Statement

- **Components of `for` statements are not mandatory.** optional
  - Initialization can be omitted if we use already declared variables for the condition
  - Loop-continuation condition can be omitted in two or more cases. When it is omitted, C assumes that the condition is always true.
    - The condition is checked inside of the body with `break` and/or `continue`.
    - The `for` statement intentionally runs infinitely.
  - Increment part can be omitted when the variables are updated inside of the body.

```
// an extreme case
for( ; ; ) {
        (code block)
}
```

for example (a typical one)

Control variable

```
#include <stdio.h>


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

```
0
1
2
3
4
5
6
7
8
9
```

Variations will come up with demo

Various for Statement

- **You can check methods of varying the control variable in a for statement.**
  - CHECK: What are the semicolons for?

```
// Vary the control variable from 1 to 100 in increments of 1.
for (i = 1; i <= 100; i++);

// Vary the control variable from 100 to 1 in increments of -1 (decrements of 1).
for (i = 100; i >= 1; --i);

// Vary the control variable from 7 to 77 in steps of 7.
for (i = 7; i <= 77; i += 7);

// Vary the control variable from 20 to 2 in steps of -2.
for (i = 20; i >= 2; i -= 2);

// Vary the control variable over the following sequence of values: 2, 5, 8, 11, 14, 17.
for (j = 2; j <= 17; j += 3);

// Vary the control variable over the following sequence of values: 44, 33, 22, 11, 0.
for (j = 44; j >= 0; j -= 11);
```

continue **with** for

- continue **continues loops.**
  - With continue, loop continues next iteration ignoring following code in the code block.
  - In for statements, the increment expression is executed, then the loop-continuation test is evaluated.

```
#include <stdio.h>
int main()
{
      int num = 0, sum = 0;
      for(int i = 0; i < 10; i++) {
            if( i % 3 == 0 )
                  continue;
            printf("%d ", i);
      }
      printf("\nContinue is used to skip printing multiples of 3.\n");
}
```

1, 2, 4, 5, 7, 8

Exercise – Summing the Evens

- Summing the Even Integers from 2 to 100

```
  (intentionally blank box)
```

```
2550
```

Exercise – Summing the Evens

- **Summing the Even Integers from 2 to 100**

```
#include <stdio.h>
int main() {
   int sum = 0;
   for (int num = 2; num <= 100; num += 2) {
      sum += num;
   }
   printf("Sum: %d\n", sum);
   return 0;
}
```

num = num + 2

```
2550
```

What we have learned …

- **Program control:** order of statements for program execution

- **Sequential execution**
  - Line-by-line execution

- **Transfer of program control**
  - Execution order is **different** from what you can see in the code.
  - Selection structure: if, if else, switch
  - Iteration structure: while, do…while, for

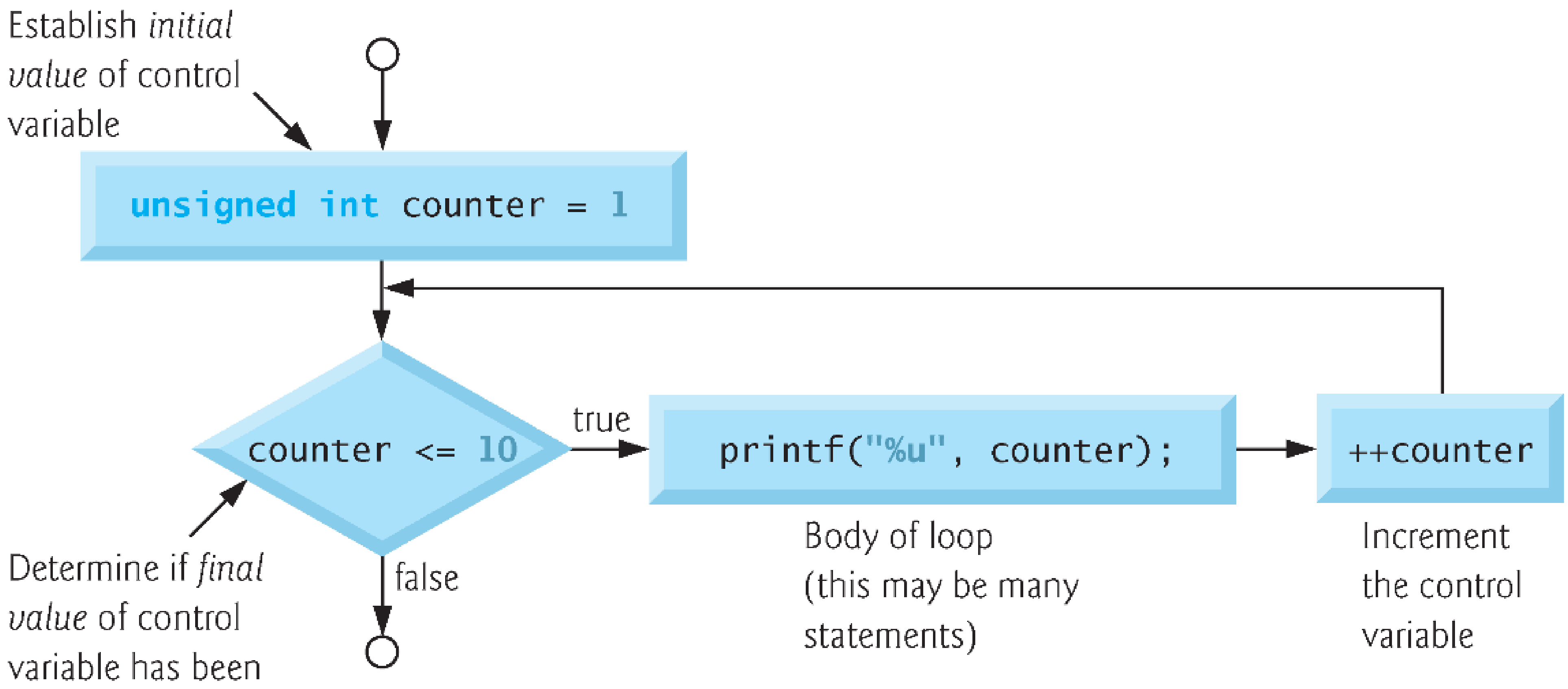
- **C has only seven control statements.**
  - And we learned all of them.

General Format of a `for` Statement

- If the condition expression is omitted, C assumes that the condition is true, thus creating an infinite loop. You may omit the initialization expression if the control variable is initialized elsewhere in the program. The increment may be omitted if it's calculated by statements in the body of the for statement or if no increment is needed. This flowchart makes it clear that the initialization occurs only once and that incrementing occurs after the body statement is performed.

```
for (unsinged int counter = 1; counter <= 10; ++counter)
   printf("%u", counter);
```

Reading Materials



Establish *initial value* of control variable

```
unsigned int counter = 1
```

```
counter <= 10
```

true

```
printf("%u", counter);
```

```
++counter
```

false

Determine if *final value* of control variable has been

Body of loop (this may be many statements)

Increment the control variable

Demo for Recap

- **Make a program that**
  - receives a number from a user
  - and prints out all the divisors of the input

약수

```
(intentionally blank box)
```

Show divisors

```c
#include <stdio.h>

int main()
{
  int number = 0;
  printf(“Enter a number: ");
  scanf("%d", &number);

  for(int i = 1; i <= number; i++) {
    if(number % i == 0) {
      printf("%d ", i);
    }
  }

  printf("\n");
}
```

Hands-on Lab

- for.c
- for_continue.c
- even_sum.c
- factorial.c
- divisors.c

[DASF004-42]

Basis and Practice in Programming

(프로그래밍 기초와 실습)

Spring 2022

HYUNGJOON KOO

SUNG KYUN KWAN UNIVERSITY(SKKU)

getchar and puts

Characters

- Characters can be stored in any integer data type because they're usually represented as one-byte integers in the computer.
  - Although they are normally stored in variables of type char, ...
- Thus, we can treat a character as either an integer or a character, depending on its use.
- When assigning a character literal to an integer variable, 'promotion' happens regarding the data type.
  - From 1 byte to 4 bytes

Casting

```
int c = 'a';            // OK!!!
printf("%c\n", c);      // Ok for this case because we know the value of c
                        // Some concerns about loosing data still exists.
return 0;
```

```
a
```

Characters

specifier

```
printf("The character (%c) has the value %d.\n", 'a', 'a');
```

```
The character (a) has the value 97.
```

- The integer 97 is the character’s numerical representation in the computer.
- Many computers today use the **ASCII (American Standard Code for Information Interchange) character set** in which 97 represents the lowercase letter 'a'.

getchar

- The getchar function (from <stdio.h>) reads one character from the keyboard and returns as an int the character that the user entered.
  - Syntax: int getchar (void);
  - getchar() return the character read as an unsigned char cast to an int or EOF on end of file or error.
  - Considering the promotion issue, it is recommended to assign the returned values of this function to an integer type variable.

#include <stdio.h>

```
int c;                       // or, 'char c;' to use the char data type.
c = getchar();               // When you use 'char', implicit type casting happens.
printf("%c\n", c);
```

(암묵적) 형변환

int a = (int) 'c';

explicit (명시적)

Characters

- **A simple example for characters**

  - Variations on data types will be shown.

```
#include <stdio.h>
int main() {
  int c = 'a';
  printf("%c\n", c);

  c = getchar();
  printf("The character (%c) has the value %d.\n", c, c);
  return 0;
}
```

char type

'b'

b

98

Reading Character Input

```
#include <stdio.h>

int main()
{
 int c;
 int num_of_a = 0;

 while( (c = getchar()) != EOF ) {
  if(c == 'a') {
   num_of_a++;
  }
 }
 printf("The number of 'a' is %d.\n",
     num_of_a);

 return 0;
}
```

- This program counts the number of 'a' from the user input.
  - The parenthesized assignment (`c = getchar()`) executes first.
  - In the program, the value of the assignment `c = getchar()` is compared with the value of EOF (a symbol whose acronym stands for "end of file").
  - We use EOF (which normally has the value -1) as the sentinel value.

Reading Character Input

```c
#include <stdio.h>

int main()
{
  int c;
  int num_of_a = 0;

  while( (c = getchar()) != EOF ) {
    if(c == 'a') {
      num_of_a++;
    }
  }
  printf("The number of 'a' is %d.\n",
              num_of_a);

  return 0;
}
```

'\n'

newline

2

aabb↵

- This program counts the number of 'a' from the user input.
  - End-of-file (EOF): A system-dependent keystroke combination to mean "end of file"—i.e., “I have no more data to enter.”
  - If the value entered by the user is equal to EOF, the while loop terminates.
  - The variable c is declared as `int` because EOF has an integer value (normally -1).

linux/unix ctrl+D

Entering the EOF Indicator

Entering the EOF Indicator

*NIX

- On Linux/UNIX/Mac OS X systems, the EOF indicator is entered by typing

```
<Ctrl> + d
```

- This notation <Ctrl> d means to press the Enter key then simultaneously press both Ctrl and d.

- On other systems, such as Microsoft Windows, the EOF indicator can be entered by typing

```
<Ctrl> + z
```

- You may also need to press *Enter* on Windows.

Reading Character Input

SUNG KYUN KWAN UNIVERSITY(SKKU)

- **This program counts the number of 'a' from the user input.**
  - What happens when we use 'char' data type for the variable c? Is it OK?
  - Check the difference when we use '\n' instead of EOF. (Ex: Counting the number of '\n'.)

```
#include <stdio.h>
int main() {
  int c;
  int num_of_a = 0;
  while( (c = getchar()) != EOF ) {
    if(c == 'a') { num_of_a++; }
  }
  printf("The number of 'a' is %d.\n", num_of_a);
  return 0;
}
```

puts

<stdio.h>

- Function puts takes a string as an argument and displays the string followed by a **newline character**.
  - Syntax: `int puts ( const char * str );`
  - Similar to `printf` but only for C string

```
printf("Hi, this statement is using printf\n");
puts("Hi, I'm using puts instead of printf");     // Check that puts has no \n.
```

Exercise: ASCII Printer for characters

- **Make a program that prints ASCII values for user-entered characters.**
  - You must print ASCII values only for characters.
  - The program terminates when EOF is entered.

'\n'

```
#include <stdio.h>

int main() {
  int c;

  /* Fill in here */

  return 0;
}
```

Exercise: ASCII Printer for characters

- **Make a program that prints ASCII values for user-entered characters.**
  - You must print ASCII values only for characters.
  - The program terminates when EOF is entered.

```
#include <stdio.h>


int main() {
  int c;
  while( (c = getchar()) != EOF) {
    if((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z')) {
      printf("Character: %c, ASCII value: %d\n", c, c);
    }
  }
  return 0;
}
```

('\n')

[a-z A-Z]

Hands-on Lab

- getchar.c
- getchar2.c
- ascii.c