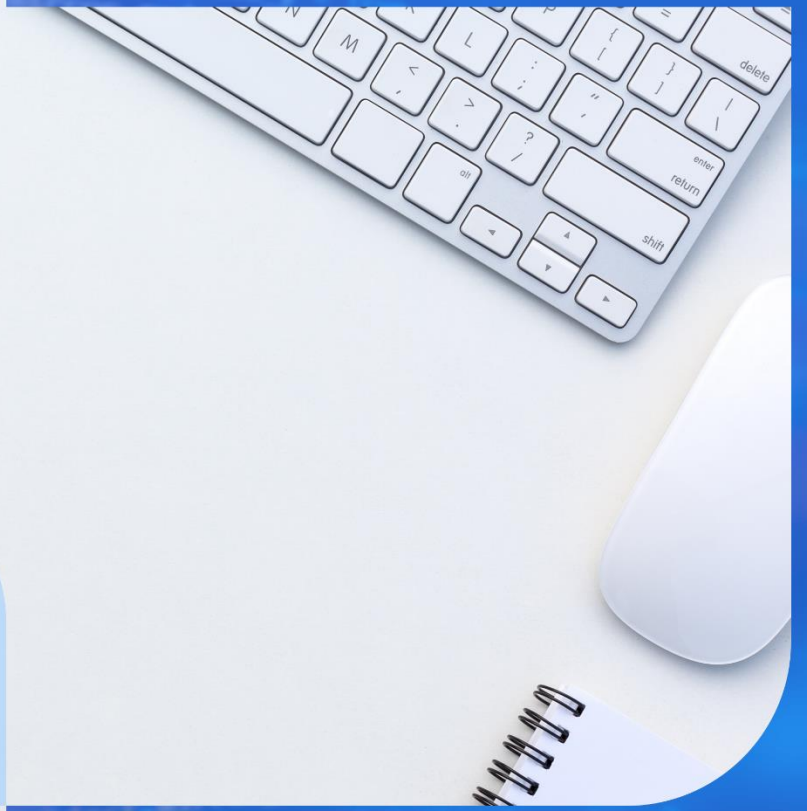
Basis and Practice in Programming

SUNG KYUN KWAN UNIVERSITY(SKKU)

Programming Basics 2



Instructor Younghoon Kim

Control Structures

- **Program control**
  - Specifying the order of statements for program execution
- **Sequential execution**
  - 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**
  - The order in which individual statements are executed or evaluated.

- **Three types of control structures**
  - Sequence structure
  - Selection structure
    - » if, if…else, switch
  - Iteration structure
    - » while, do…while, for
  - C has only seven control statements.

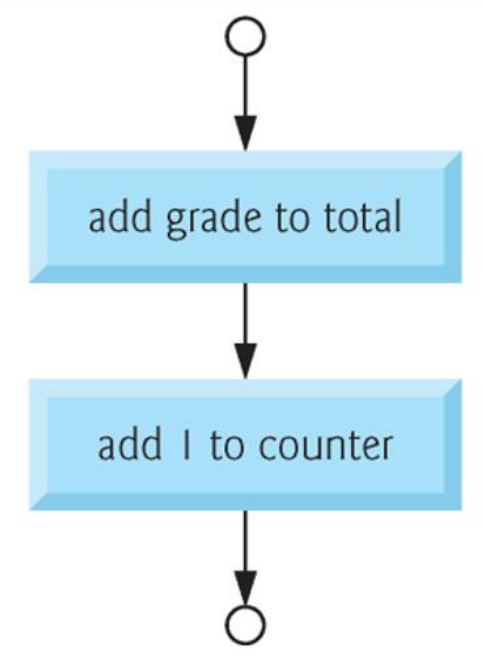
Some terms

- **Algorithm**
  - a procedure for solving a problem in terms of actions and order
- **Pseudocode**
  - an artificial and informal language that helps you develop algorithms
  - actions and decision statements written in normal language
- **Flowchart**
  - a graphical representation of an algorithm or of a portion of an algorithm

Examples with Sequence Structure

SUNG KYUN KWAN UNIVERSITY(SKKU)

- **Flowchart**



- **Pseudocode**

```
add grade to total
increase counter by 1
```

- **C code**

```
total = total + grade;
counter = counter + 1;
```

Algorithms

- The solution to any computing problem involves executing a series of actions in a specific order. Algorithm is a procedure for solving a problem in terms of the actions to be executed and the order in which these actions are to be executed. Specifying the order in which statements are to be executed in a computer program is called program control.

Pseudocode

- Pseudocode is an artificial and informal language that helps you develop algorithms. Pseudocode is similar to everyday English; it’s convenient and user friendly although it’s not an actual computer programming language. Pseudocode consists only of action and decision statements—those that are executed when the program has been converted from pseudocode to C and is run in C.

Flowcharts

- A flowchart is a graphical representation of an algorithm or of a portion of an algorithm. We use the rectangle symbol, also called the action symbol, to indicate any type of action including a calculation or an input/output operation. When drawing a flowchart that represents a complete algorithm, a rounded rectangle symbol containing the word "Begin" is the first symbol used in the flowchart; an oval symbol containing the word "End" is the last symbol used. When drawing only a portion of an algorithm as in our example, the rounded rectangle symbols are omitted in favor of using small circle symbols, also called connector symbols. Perhaps the most important flowcharting symbol is the **diamond symbol**, also called the decision symbol, which indicates that a decision is to be made.

Control Structures

- Normally, statements in a program are executed one after the other in the order in which they're written. This is called **sequential execution**. Various C statements we'll soon discuss enable you to specify that the next statement to be executed may be other than the next one in sequence. This is called **transfer of control**. Research had demonstrated that all programs could be written in terms of only three control structures, namely the **sequence structure**, the **selection structure** and the **iteration structure**. The sequence structure is simple—unless directed otherwise, the computer executes C statements one after the other in the order in which they're written.

Selection Statements in C

- C provides three types of selection structures in the form of statements.
- The **if** selection statement either selects (performs) an action if a condition is true or skips the action if the condition is false. The **if** statement is called a single-selection statement because it selects or ignores a single action.
- The **if…else** selection statement performs an action if a condition is true and performs a different action if the condition is false. The **if…else** statement is called a double-selection statement because it selects between two different actions.
- The **switch** selection statement performs one of many different actions depending on the value of an expression. The switch statement is called a multiple-selection statement because it selects among many different actions.

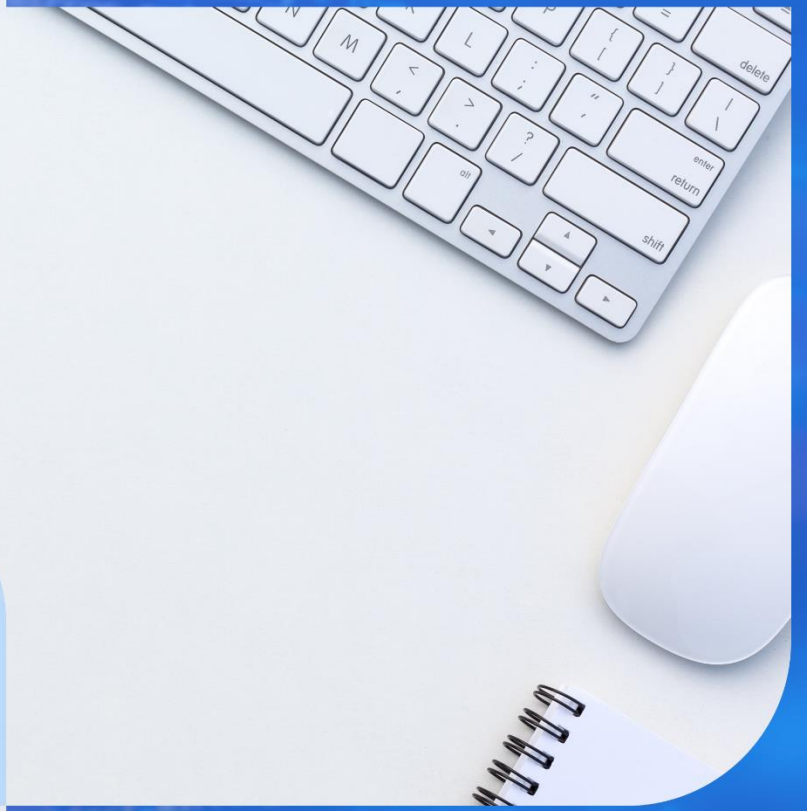
Iteration Statements in C

- C provides three types of iteration structures in the form of statements, namely **while**, **do…while**, and **for**.

Basis and Practice in Programming

SUNG KYUN KWAN UNIVERSITY(SKKU)

Programming Basics 2



Instructor Younghoon Kim

`if` Statement

Logical Expressions

SUNG KYUN KWAN UNIVERSITY(SKKU)

- **Expressions that can be judged as true or false.**
  - ex1) i = 10; // We cannot determine true or false
  - ex2) i <= j // With valid values of i and j, it can be true or false.
  - In C, **zero** is considered as **false**, and **non-zeroes** are as **true**.

- **Equality and Relational Operations**
  - Operations that are used for logical expressions
  - ==, !=, < , >, <=, >=
  - Left associative
    - The operations are grouped from the left
    - Think about ‘i < j < k ‘

- **Logical Operators**
  - Operators such as *negation*, '*and*' and '*or*'
  - !, &&, ||

Equality and Relational Operators

SUNG KYUN KWAN UNIVERSITY(SKKU)

Algebraic equality or relational operator	C equality or relational operator	Example of C condition	Meaning of C condition
Relational operators			
>	>	x > y	x is greater than y
<	<	x < y	x is less than y
≥	>=	x >= y	x is greater than or equal to y
≤	<=	x <= y	x is less than or equal to y
Equality operators			
=	==	x == y	x is equal to y
≠	!=	x != y	x is not equal to y

if statement

SUNG KYUN KWAN UNIVERSITY(SKKU)

- One of selection statements (a.k.a. branches)
- Syntax

```
if(logical_expression_1) {
     /* When logical_expression_1 is true */
     (statements)
} else if (logical_expression_2) {
     /* Optional. When logical_expression_1 is false
        and logical_expression_2 is true */
     (statements)
} else {
     /* Optional. When logical_expression_1 and
         logical_expression_2 are false */
     (statements)
}
```

{, } can be omitted with a single statement in a block.

if statement

SUNG KYUN KWAN UNIVERSITY(SKKU)

```
if(logical_expression_1) {
      /* When logical_expression_1 is true */
      (statements)
}
```

- **Logical expressions are called conditions for `if` statements.**
  - When the condition is true, statements in the corresponding block({, }) are executed.
  - When it is false, the body is not executed.
  - Check the indentation. It is used for better readability.

`if` statement Example - I

SUNG KYUN KWAN UNIVERSITY(SKKU)

```
#include <stdio.h>
int main() {
    int i = 0;
    if (i == 0)
        printf("i is zero.\n");
    return 0;
}
```

```
i is zero.
```

1. The `if` statement compares the value of a variable `i` and `0` to test for equality. (It is true.)
2. The corresponding body statement, `printf("i is zero.\n")`, is executed.
   (Braces can be omitted with one line in the body.)

- Indentation enhances *readability*.
- What happens when `i` is not `0`?

Common mistakes for beginners

DEMO

```c
#include <stdio.h>
int main() {
    int i = 0;
    if (i = 0)
        printf("i is zero.\n");
    return 0;
}
```

```c
#include <stdio.h>
int main() {
    int i = 0;
    if (i = 0);
    printf("i is zero.\n");
    return 0;
}
```

- **Above two programs have logical errors.**
  - One type of runtime errors, hard to debug
- **Where are they?**

if…else statement

SUNG KYUN KWAN UNIVERSITY(SKKU)

```
if(logical_expression_1) {
     (statements)
} else {
     /* Optional. When logical_expression_1 is false */
     (statements)
}
```

- **To define actions when the condition for `if` is false.**
- **Double-selection statement**
  - Vs. single-selection statement (`if` statement without else)
- **When `logical_expression_1` is**
  - true, `if` block is executed.
  - false, `else` block is executed.

`if` statement Example - II

DEMO

```
#include <stdio.h>
int main() {
    int i = 1;
    if (i == 0) printf("i is zero.\n"); // Shrinking code lines
    else printf("i is NOT zero.\n);
    return 0;
}
```

```
i is NOT zero.
```

1. The `if` statement compares the value of a variable `i` and `0` to test for equality. (It is false.)
2. The body statement for else, `printf("i is NOT zero.\n")`, is executed.

- We can reduce the lines of code by placing statements in one line.
  - Not recommended for beginners
- Think about the concepts of program control with this example.

else if block


```
if(logical_expression_1) {
     (statements)
} else if (logical_expression_2) {
     /* Optional. When logical_expression_1 is false
        and logical_expression_2 is true */
    (statements)
}
```

- **To test multiple conditions**
- **The condition for `else if` is checked only when `logical_expression_1` is false.**
  - If it is true, only the body for the first `if` will be executed.
- **With false conditions for both nothing will happen with the above code.**

if statement Example - III

SUNG KYUN KWAN UNIVERSITY(SKKU)

```
#include <stdio.h>
int main() {
    int i = 1;
    if (i == 0) printf("i is zero.\n");
    else if (i == 1) printf("i is one.\n);
    return 0;
}
```

```
i is one.
```

1. The `if` statement compares the value of a variable `i` and 0 to test for equality. (It is false.)
2. Second condition is checked. (It is true.) The body statement for `else`, `printf("i is one.\n")`, is executed.

- What happens with `"int i  = 0;"` or `"int i = 2;"`?

`if` Statement Example - IV

DEMO

- **This program prints a battery-level color based on the user input.**
  - Variation: We will add 'Black level' where the battery level is zero. Where to add it?

```
#include <stdio.h>
int main() {
    double battery_level;
    printf("Enter your battery level: ");
    scanf("%lf", &battery_level);
    if (battery_level < 20) printf("RED LEVEL. NEED CHARGING!\n");
    else if (battery_level >= 50) printf("Green level. OK!\n");
    else printf("Yellow level. May need charging soon.\n");
    return 0;
}
```

`if` Statement Example - IV


```
if (battery_level < 20) printf("RED LEVEL. NEED CHARGING!\n");
else if (battery_level >= 50) printf("Green level. OK!\n");
else printf("Yellow level. May need charging soon.\n");
```

`battery_level` is a variable and determined on runtime.

Nested if

- `if` statement is a statements.
- Any statements can be placed in a block.
  → `if` statements can be placed in `if…else` blocks
- It is called **'nested if'**.

Example cases

- Checking whether a number is a multiple of two but not of three
- Determine whether a flag has both colors of blue and red
- Finding maximum among three numbers (See the next example code.)

`if` Statement Example - V (nested if)

DEMO

Remark: **`if`** statement is treated as one statements with or without **`else if`** or **`else`** blocks.

```
#include <stdio.h>
int main() {
     int max, i = 4, j = 10, k = 1;       // declaring multiple variables
     if (i > j) {
          if (i > k) max = i;             // if block in if block
          else max = k;
     } else {
          if (j > k) max = j;             // if block in if block
          else max = k;
     }
     printf("max: %d\n", max);
     return 0;
}
```

```
max: 10
```

Exercise

DEMO

- **even/odd checker**
  - Make a program that receives a number from users and determines whether the input is odd or even

```
(intentionally blank box)
```

even/odd checker

- Make a program that receives a number from users and determines whether the input is odd or even


```
// This program accepts an integer from a user
// and checks whether it is odd or even.
#include <stdio.h>

int main()
{
        int number;
        printf("Input a number : ");
        scanf("%d", &number);

        if (number % 2 == 0)
                printf("%d is even.\n", number);
        else
                printf("%d is odd\n", number);
        return 0;
}
```

Decision making

- C's if statement allows a program to make a decision based on the truth or falsity of a statement of fact called a condition. If the condition is true (i.e., the condition is met) the statement in the body of the if statement is executed. If the condition is false (i.e., the condition isn't met) the body statement is not executed. Whether the body statement is executed or not, after the if statement completes, execution proceeds with the next statement after the if statement. Conditions in if statements are formed by using the equality operators and relational operators.

The `if` Selection Statement

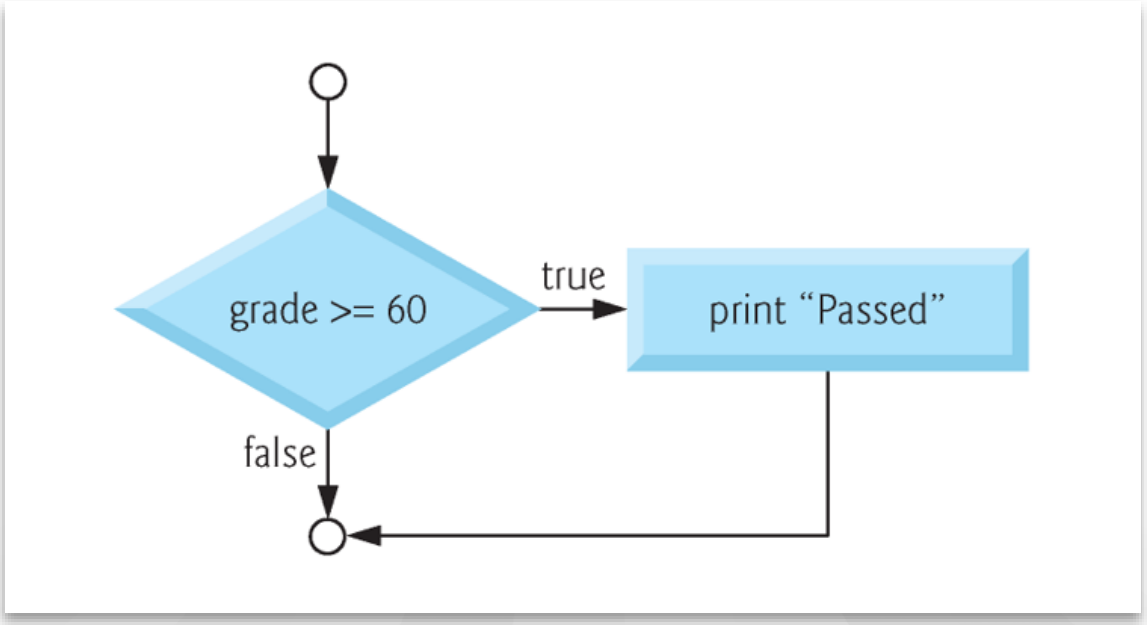
- Selection statements are used to choose among alternative courses of action. The below pseudocode statement determines whether the condition "student's grade is greater than or equal to 60" is true or false. If the condition is true, then "Passed" is printed, and the next pseudocode statement in order is "performed". If the condition is false, the printing is ignored, and the next pseudocode statement in order is performed.

```
If student's grade is greater than or equal to 60
     Print "Passed"
```

- This pseudocode If statement may be written in C as

```
if ( grade >= 60 ) {
   printf( "Passed\n" );
} // end if
```

- **Flowchart for the previous example**
  - The flowchart illustrates the single-selection if statement. The diamond symbol (also called the decision symbol) indicates that a decision is to be made. The decision symbol contains an expression, such as a condition, that can be either true or false.



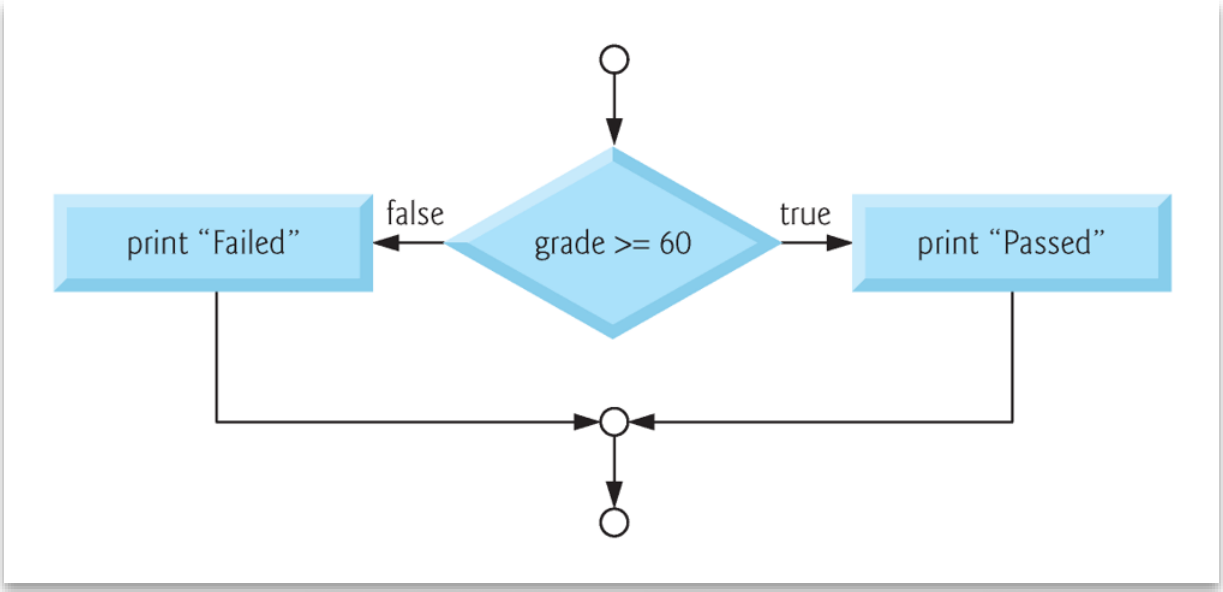
The `if...else` Selection Statement

- The `if…else` selection statement allows you to specify that different actions are to be performed when the condition is true and when it's false. For example, the below pseudocode statement prints `Passed` if the student's grade is greater than or equal to 60 and `Failed` if the student's grade is less than 60. In either case, after printing occurs, the next pseudocode statement in sequence is "performed." The body of the else is also indented.

```
If student's grade is greater than or equal to 60
     Print "Passed"
else
     Print "Failed"
```

- **C code and flowchart for the previous example**
  - The previous pseudocode `if...else` statement may be written in C as below and the flowchart for it is also shown below.

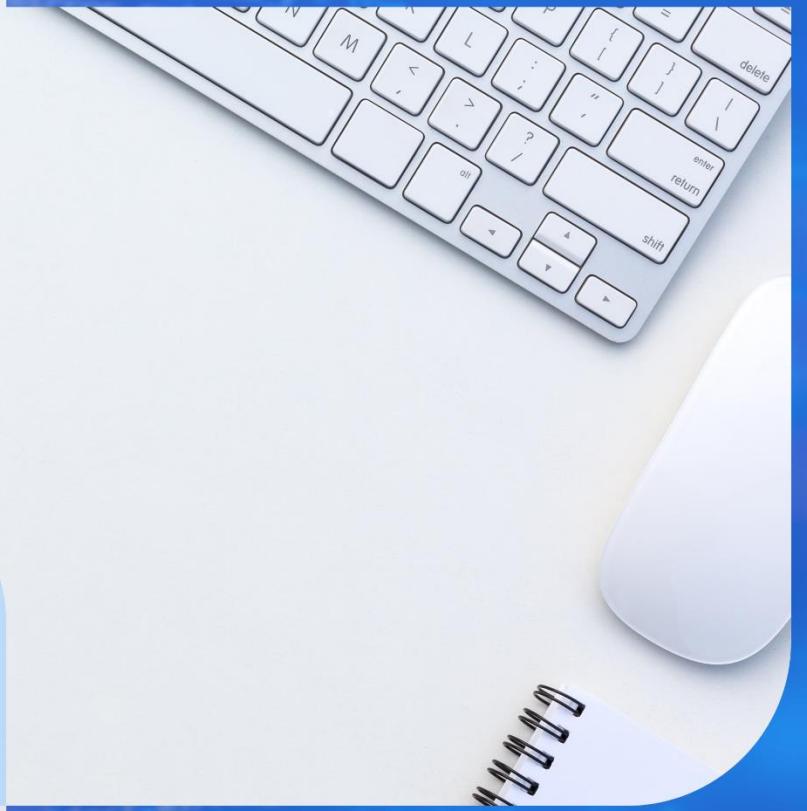
```
if ( grade >= 60 ) {
   printf( "Passed\n" );
} // end if
else {
   printf( "Failed\n" );
} // end else
```



Basis and Practice in Programming

SUNG KYUN KWAN UNIVERSITY(SKKU)

Programming Basics 2



Instructor Younghoon Kim

`switch` Statement

- **Or, simply, `switch` statements**
- **One of selection statements (a.k.a. branches)**
- **When do we use switch-case?**
  - A series of decisions on integers or on expressions evaluated into integer values

```
if(a == 1)
     printf("one\n");
else if(a == 2)
     printf("two\n");
else if(a == 3)
     printf("three\n");
else if(a == 4)
     printf("four\n");
else
     printf("Illigal\n");
```

Syntax of `switch-case`

SUNG KYUN KWAN UNIVERSITY(SKKU)

- Syntax:

```
switch(controlling_expr) {
      case constant_expr_1:          // --> the first case label
            code_block_1
      case constant_expr_2:          // --> the second case label
            code_block_2
      …
      default:
            code_block_default
}
```

- `switch` statement starts with a variable name in parentheses
  - Controlling expression (should be integers or reduced to integers)
- The value of controlling expression is compared with each of constant expression.
  - Case labels (should be integers or reduced to integers)
- If a match occurs, the statements for that case are executed.
- If no match occurs, the default case is executed.

Syntax of `switch-case`

SUNG KYUN KWAN UNIVERSITY(SKKU)

- Syntax:

```
switch(controlling_expr) {
      case constant_expr_1:          // --> the first case label
            code_block_1
      case constant_expr_2:          // --> the second case label
            code_block_2
      …
      default:
            code_block_default
}
```

- A code block for each case usually ends with break.
  - Without breaks, the program continues executing across cases.
- Useful when branching on constant integer values.
- The constant expressions must be evaluated into an integer.
  - characters are acceptable.
- Default statement is optional.

- `break`: **Explicitly managing program control**
- **Two usages**
  - With loops: Immediately termination of a loop.
    - The program control resumes at the next statement following the loop
    - Will be covered later
  - With switch: Termination of a case
    - The break statement causes program control to continue with the first statement after the switch statement.
    - Without breaks, the program continues executing across cases.
    - Will be checked with demo

switch-case syntax with examples

SUNG KYUN KWAN UNIVERSITY(SKKU)

```c
#include <stdio.h>
int main() {
    int num = 8;
    switch (num) {
        case 7:
            printf("Value is 7\n");
            break;
        case 8:
            printf("Value is 8\n");
            break;
        case 9:
            printf("Value is 9\n");
            break;
        default:
            printf("Out of range\n");
    }
    printf("End of switch\n");
    return 0;
}
```

- **A code block for each case usually ends with break.**
  - The last case does not need it.
    (It is default case in this example.)
  - The next statement for execution is the one with "End of switch".
- **The constant expressions must be evaluated into an integer.**
  - If they are not, errors will be shown on compile-time.

switch-case syntax with examples


```c
#include <stdio.h>
int main() {
    int num = 8;
    switch (num) {
        case 7:
        case 8:
            printf("Value is 7 or 8");
        case 9:
            printf("Value is 9");
        default:
            printf("Out of range");
    }
    printf("End of switch\n");
    return 0;
}
```

- **Without breaks, the program continues executing across cases.**

Grading program with a switch statement

DEMO

```c
#include <stdio.h>

int main()
{
  int grade = 0;
  printf("Enter a grade(0~4): ");
  scanf("%d", &grade);
  switch (grade) {
    case 4:
      printf("Excellent");
      break;
    case 3:
      printf("Good");
      break;
    case 2:
      printf("Average");
      break;
    case 1:
      printf("Poor");
      break;
    case 0:
      printf("Fail");
      break;
    default:
      printf("Illegal grade");
  }
  printf("\n");

  return 0;
}
```

Pass/Fail program with a switch statement

DEMO

```c
#include <stdio.h>

int main()
{
  int grade = 0;
  printf("Enter a grade(0~4): ");
  scanf("%d", &grade);
  switch (grade) {
    case 4:
    case 3:
    case 2:
    case 1:
      printf("Pass");
      break;
    case 0:
      printf("Fail");
      break;
    default:
      printf("Illegal grade");
      break;
  }
  printf("\n");

  return 0;
}
```

- `switch` **Multiple-Selection Statement**
  - Occasionally, an algorithm will contain a series of decisions in which a variable or expression is tested separately for each of the constant integral values it may assume, and different actions are taken. This is called multiple selection. C provides the switch multiple-selection statement to handle such decision making. The `switch` statement consists of a series of case labels, an optional default case and statements to execute for each case.

- `break` **statement**
  - The `break` statement causes program control to continue with the first statement after the `switch` statement. The `break` statement is used because the cases in a switch statement would otherwise run together. If `break` is not used anywhere in a `switch` statement, then each time a match occurs in the statement, the statements for all the remaining cases will be executed.