Introduction to Computers & Programming

Comp 1330/ First Semester 2024/2025

Instructor: Saif Harbia

Faculty of Engineering and Technology

Department of Computer Science

Chapter 04

Selection Structures:
if and switch Statements

Chapter Objectives:

1. statements that control the flow of program execution
2. conditions and logical expressions
3. use if and switch statements to select one statement group to execute from many alternatives
4. trace an algorithm or program to verify that it does what you expect.

4.1 CONTROL STRUCTURES

- **Control structures** control the flow of execution in a program or function
- They enable you to combine individual instructions into a single logical unit with one entry point and one exit point.
- Instructions are organized into three kinds of control structures to control execution flow: **sequence, selection, and repetition** (CH05)
- Until now we have been using only **sequential flow**.
- **A compound statement** (1), written as a group of statements bracketed by { and } , is used to specify sequential flow.
- **A selection control structure** chooses which alternative to execute. (2)

4.2 CONDITIONS

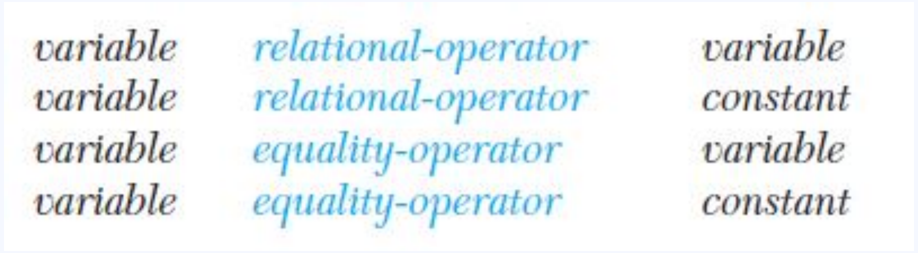
➢ A program chooses among alternative statements by testing the value of key variables

➢ If **rest_heart_rate** is a type int variable, then

rest_heart_rate > 75 (1)

➢ Such an expression is called a **condition** because it establishes a criterion for either executing or skipping a group of statements.

RELATIONAL AND EQUALITY OPERATORS



variable	*relational-operator*	*variable*
variable	*relational-operator*	*constant*
variable	*equality-operator*	*variable*
variable	*equality-operator*	*constant*

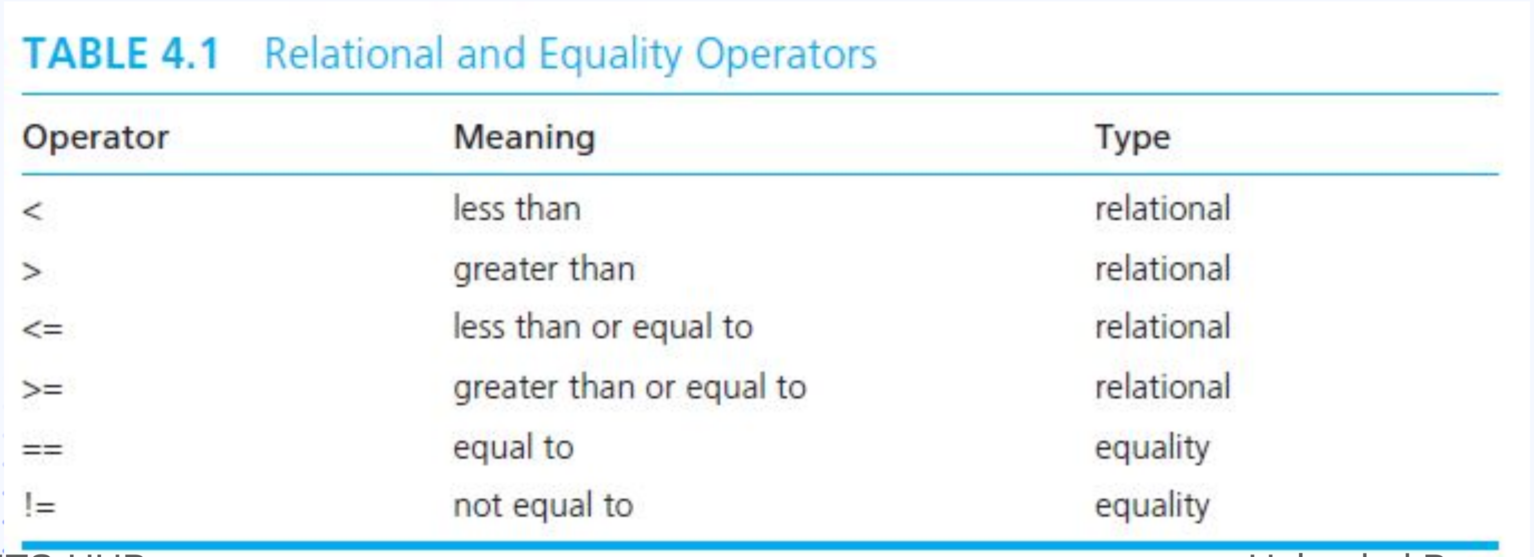
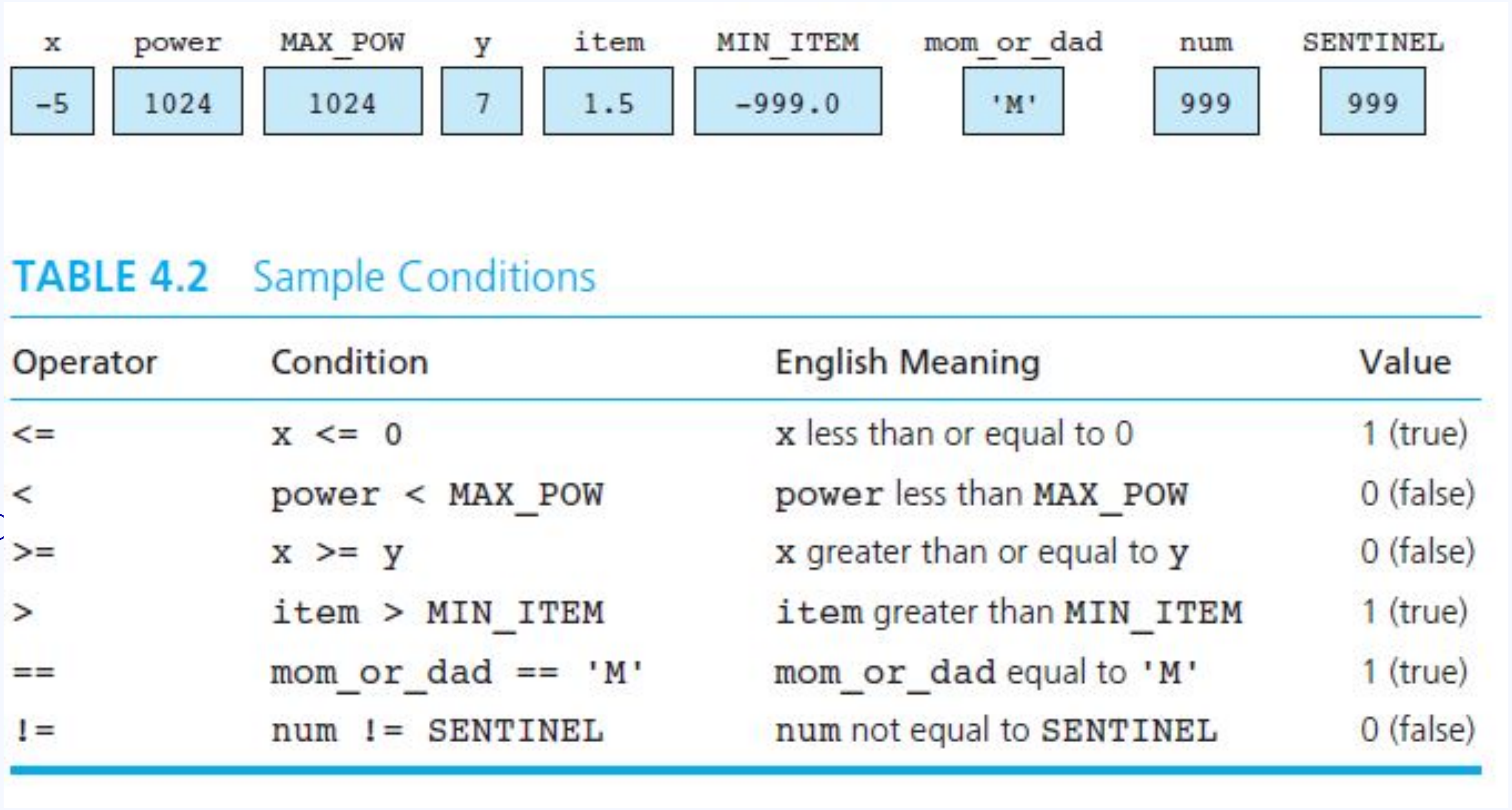


TABLE 4.1 Relational and Equality Operators

Operator	Meaning	Type
<	less than	relational
>	greater than	relational
<=	less than or equal to	relational
>=	greater than or equal to	relational
==	equal to	equality
!=	not equal to	equality



x	power	MAX_POW	y	item	MIN_ITEM	mom_or_dad	num	SENTINEL
-5	1024	1024	7	1.5	-999.0	'M'	999	999

TABLE 4.2 Sample Conditions

Operator	Condition	English Meaning	Value
`<=`	`x <= 0`	`x` less than or equal to 0	1 (true)
`<`	`power < MAX_POW`	`power` less than `MAX_POW`	0 (false)
`>=`	`x >= y`	`x` greater than or equal to `y`	0 (false)
`>`	`item > MIN_ITEM`	`item` greater than `MIN_ITEM`	1 (true)
`==`	`mom_or_dad == 'M'`	`mom_or_dad` equal to `'M'`	1 (true)
`!=`	`num != SENTINEL`	`num` not equal to `SENTINEL`	0 (false)

LOGICAL OPERATORS

➢ With the three logical operators, we can form more complicated conditions or logical expressions:

1. **&&** (and), 2. || (or) 3. **!** (not)

Examples:

- salary < MIN_SALARY || dependents > 5 (1)
- temperature > 90.0 && humidity > 0.90 (2)
- n >= 0 && n <= 100 (3)
- 0 <= n && n <= 100 (3)

LOGICAL OPERATORS

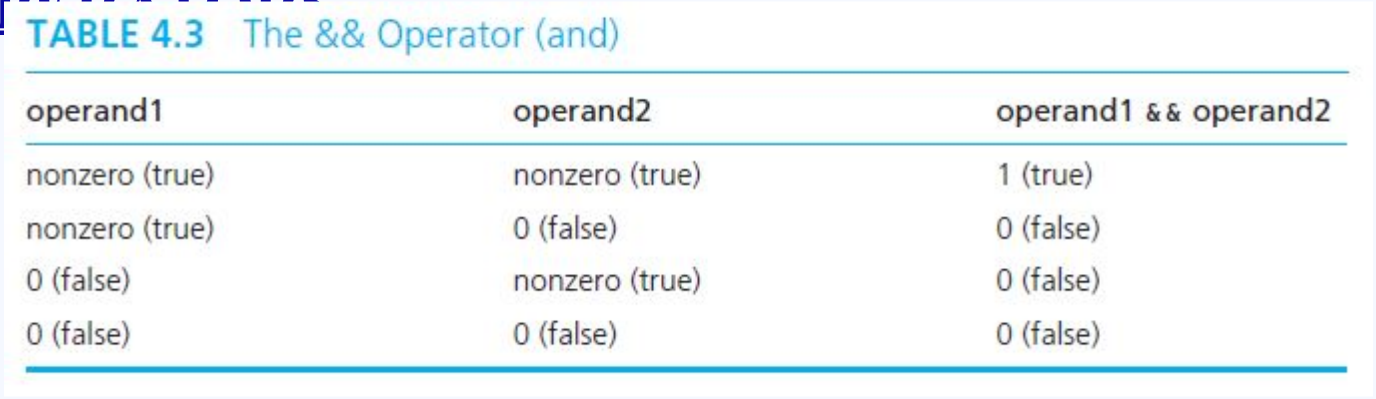


TABLE 4.3 The && Operator (and)

operand1	operand2	operand1 && operand2
nonzero (true)	nonzero (true)	1 (true)
nonzero (true)	0 (false)	0 (false)
0 (false)	nonzero (true)	0 (false)
0 (false)	0 (false)	0 (false)

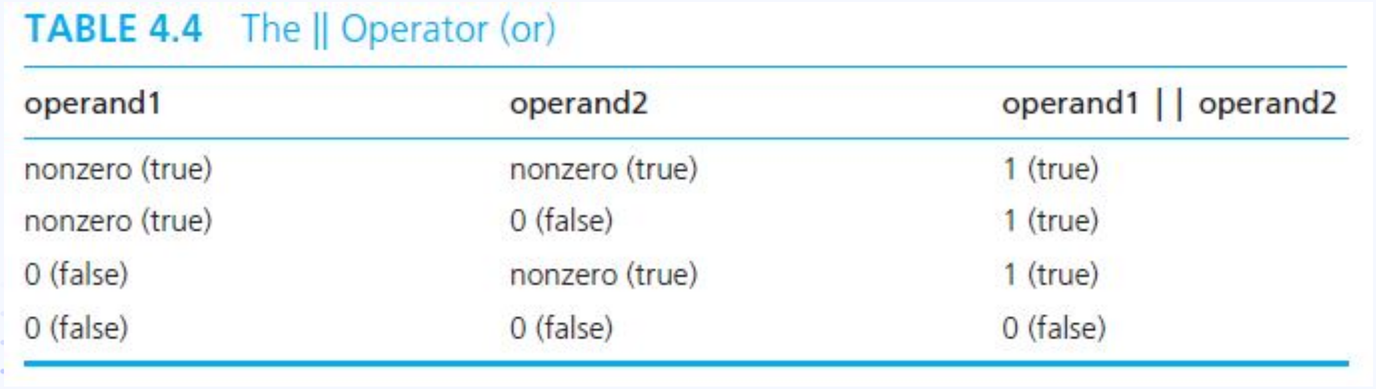


TABLE 4.4 The || Operator (or)

operand1	operand2	operand1 || operand2
nonzero (true)	nonzero (true)	1 (true)
nonzero (true)	0 (false)	1 (true)
0 (false)	nonzero (true)	1 (true)
0 (false)	0 (false)	0 (false)

LOGICAL OPERATORS

➢ The third logical operator, ! (not), has a single operand and yields the **logical complement**, or **negation** , of its operand (that is, if the variable positive is nonzero (true), !positive is 0 (false) and vice versa).

`!(0 <= n && n <= 100)` (1)

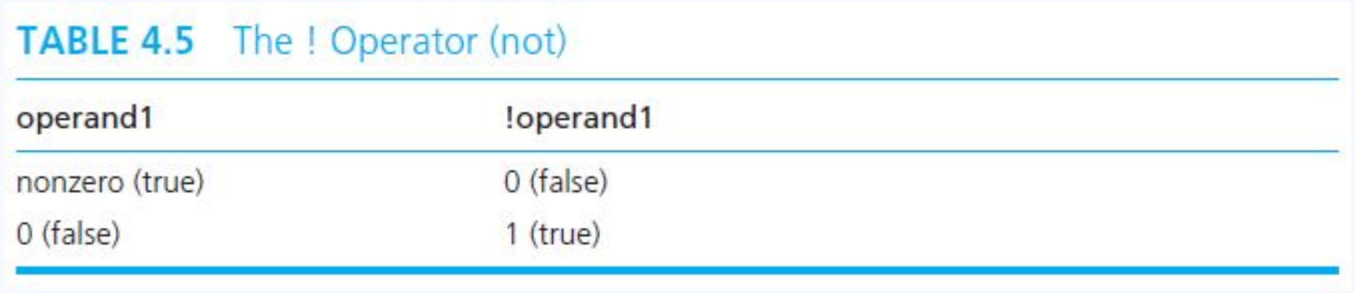


TABLE 4.5 The ! Operator (not)

operand1	!operand1
nonzero (true)	0 (false)
0 (false)	1 (true)

OPERATOR PRECEDENCE

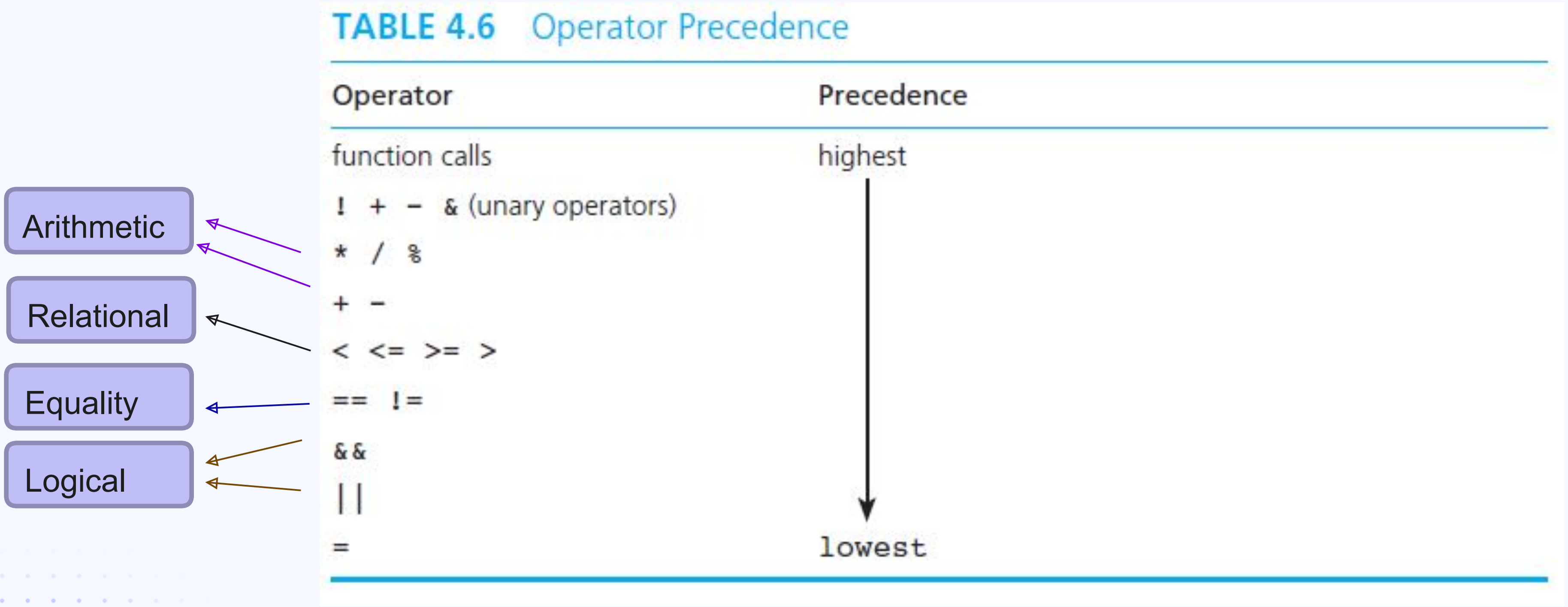

Arithmetic
Relational
Equality
Logical

TABLE 4.6 Operator Precedence

Operator	Precedence
function calls	highest
`! + - &` (unary operators)	
`* / %`	
`+ -`	
`< <= >= >`	
`== !=`	
`&&`	
`||`	
`=`	lowest

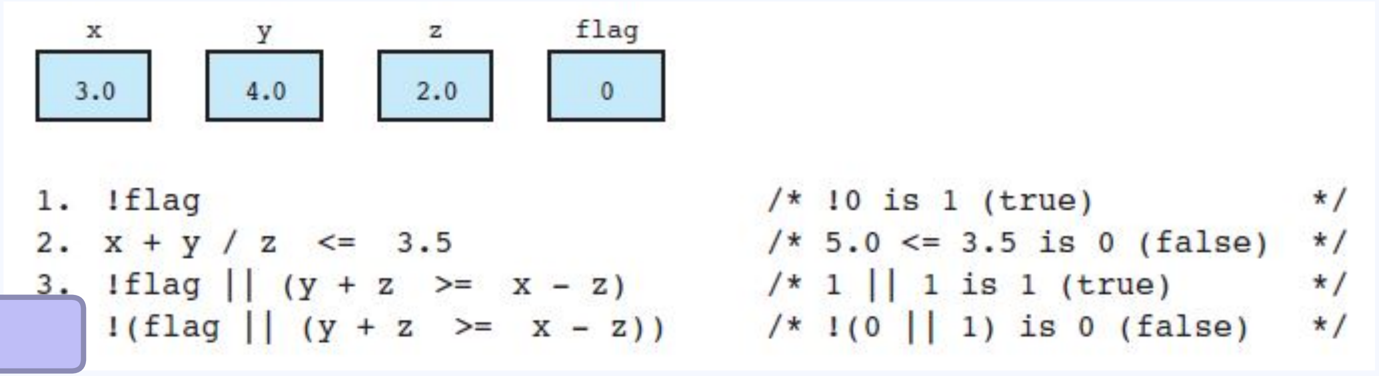
OPERATOR PRECEDENCE

- Notice that the precedence of operators + and − depends on whether they have one operand or two. In the expression **-x - y * z**
the unary minus is evaluated first (**-x**), then ***** , and then the second **-** .

- You can use parentheses to change the order of operator evaluation. In the expression **(x < y || x < z) && x > 0.0** C evaluates **II** before **&&.** (1)

- You can also use parentheses to clarify the meaning of expressions.
If x , min , and max are type double , the C compiler will interpret the expression:
x + y < min + max as **(x + y) < (min + max)** (2)

OPERATOR PRECEDENCE


x
3.0
y
4.0
z
2.0
flag
0

```
1.  !flag                          /* !0 is 1 (true)            */
2.  x + y / z  <=  3.5             /* 5.0 <= 3.5 is 0 (false)   */
3.  !flag || (y + z  >=  x - z)    /* 1 || 1 is 1 (true)        */
    !(flag || (y + z  >=  x - z))  /* !(0 || 1) is 0 (false)    */
```

Example 4.2

* Figure 4.1, p.179 shows the evaluation tree and step-by-step evaluation for expression 3.

SHORT-CIRCUIT EVALUATION

- An expression of the form **a || b** must be true if **a** is true. (1)

- Similarly, an expression of the form **a && b** must be false if **a** is false. (2)
- We can use *short-circuit evaluation* to prevent potential run-time errors. **(num % div == 0)** (3) **What if div is 0?**

- In this case, the remainder calculation would cause a division by zero run-time error.

- However, we can prevent this error by using the revised condition **([redacted]iv != 0 && (num % div == 0))** (4)

WRITING ENGLISH CONDITIONS IN C

x = 3.0 y = 4.0 z = 2.0

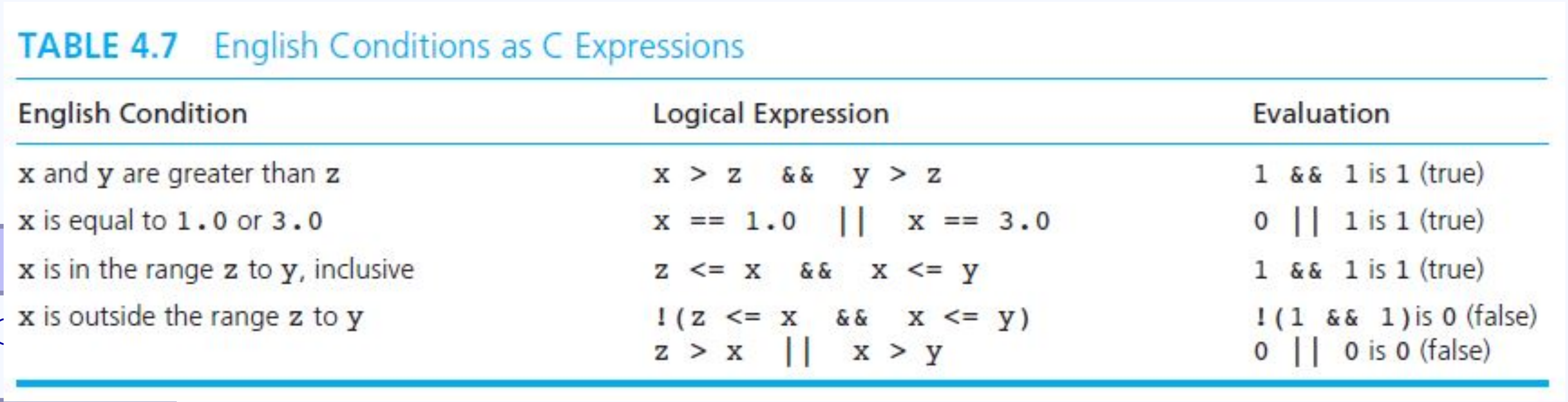


TABLE 4.7 English Conditions as C Expressions

English Condition	Logical Expression	Evaluation
`x` and `y` are greater than `z`	`x > z && y > z`	`1 && 1` is `1` (true)
`x` is equal to `1.0` or `3.0`	`x == 1.0 || x == 3.0`	`0 || 1` is `1` (true)
`x` is in the range `z` to `y`, inclusive	`z <= x && x <= y`	`1 && 1` is `1` (true)
`x` is outside the range `z` to `y`	`!(z <= x && x <= y)`	`!(1 && 1)` is `0` (false)
	`z > x || x > y`	`0 || 0` is `0` (false)

In the first logical expression You may be tempted to write this as

x && y > z /* invalid logical expression */ (1)

COMPARING CHARACTERS

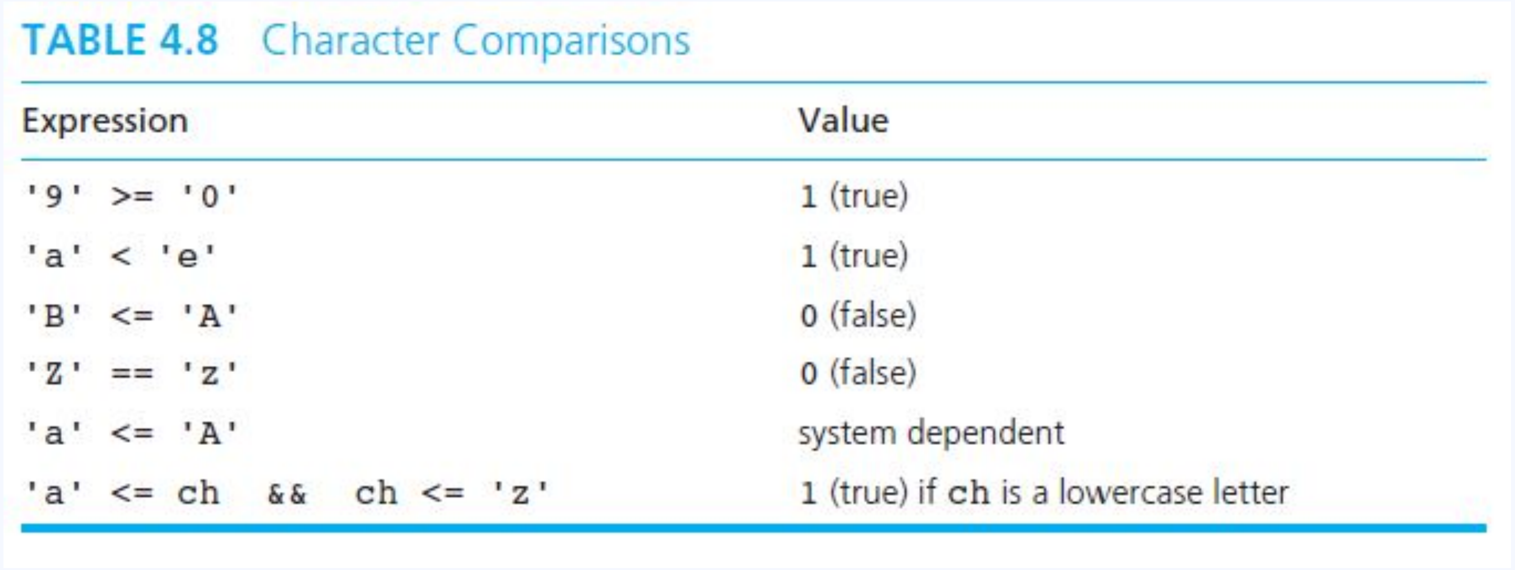
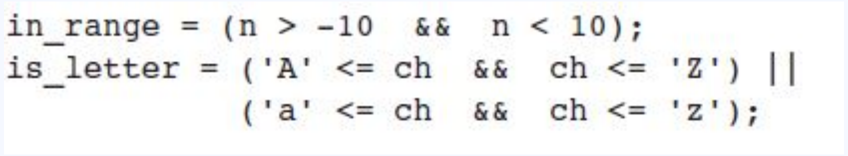


TABLE 4.8 Character Comparisons

Expression	Value
`'9' >= '0'`	1 (true)
`'a' < 'e'`	1 (true)
`'B' <= 'A'`	0 (false)
`'Z' == 'z'`	0 (false)
`'a' <= 'A'`	system dependent
`'a' <= ch && ch <= 'z'`	1 (true) if `ch` is a lowercase letter

LOGICAL ASSIGNMENT

The simplest form of a logical expression in C is a single type int value or variable intended to represent the value true or false (1)

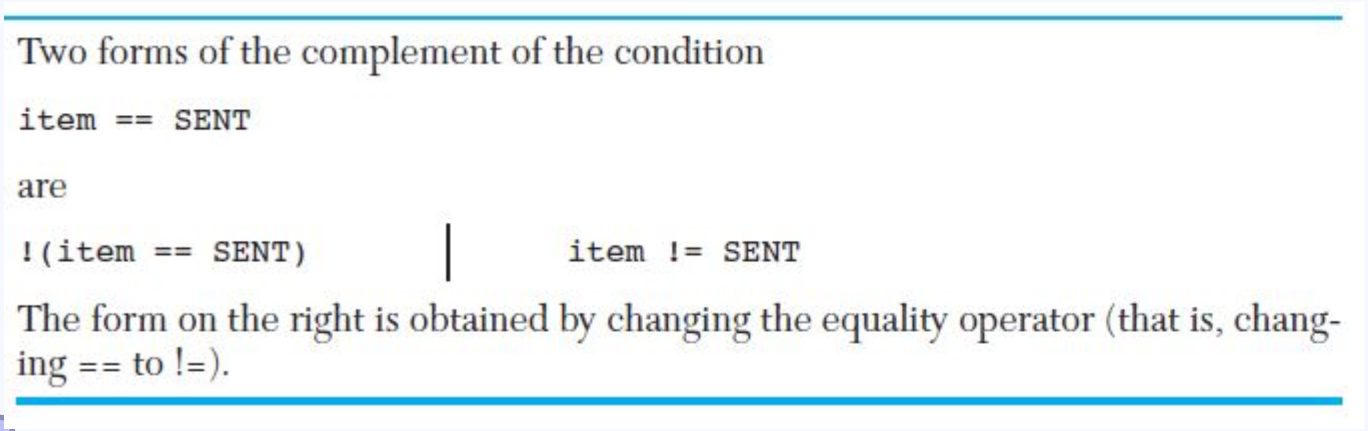


```
in_range = (n > -10  &&  n < 10);
is_letter = ('A' <= ch  &&  ch <= 'Z') ||
            ('a' <= ch  &&  ch <= 'z');
```

Example 4.5

You can delete the parentheses without affecting the order of operator evaluation.

COMPLEMENTING A CONDITION



Two forms of the complement of the condition

```
item == SENT
```

are

```
!(item == SENT)          item != SENT
```

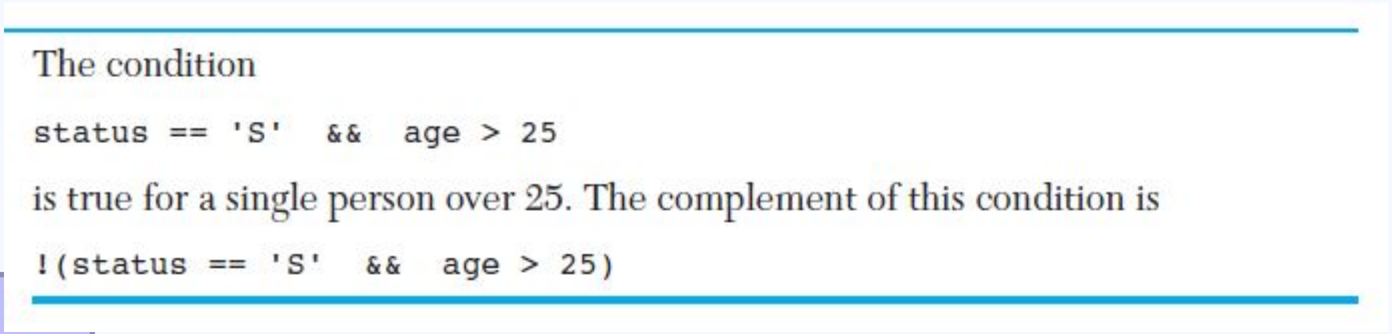
The form on the right is obtained by changing the equality operator (that is, changing == to !=).

Example 4.7

<= should be changed to >

< should be changed to >=

COMPLEMENTING A CONDITION



The condition

```
status == 'S'  &&  age > 25
```

is true for a single person over 25. The complement of this condition is

```
!(status == 'S'  &&  age > 25)
```

Example 4.8

Using **DeMorgan’s theorem**, we can write the complement of

age > 25 ~~&&~~ (status == 'S' || status == 'D')

as

age <= 25 || (status != 'S' && status != 'D') *(1)*

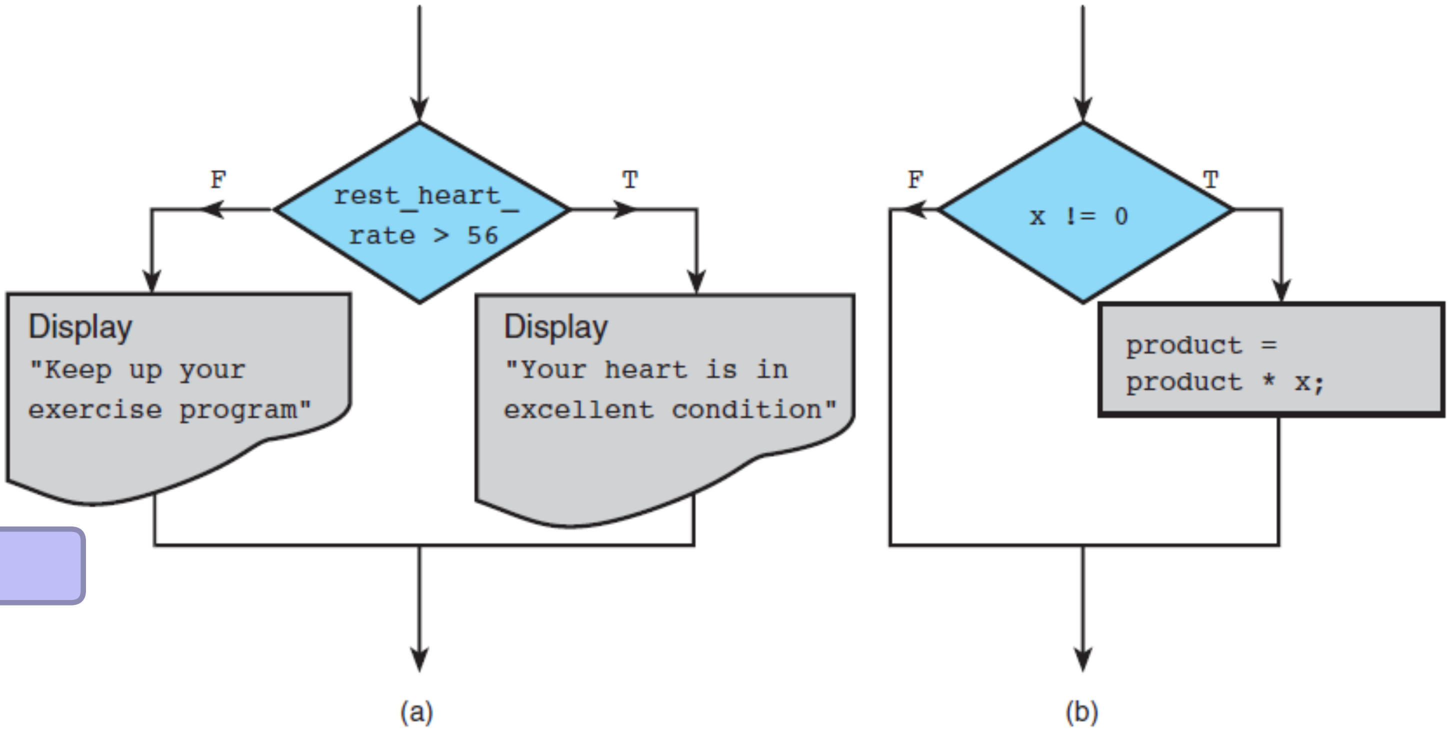
4.3 THE IF STATEMENT

- **if Statement with Two Alternatives:**

```
if (rest_heart_rate > 56)
   printf("Keep up your exercise program!\n");
else
   printf("Your heart is in excellent health!\n");
```

```
if (crsr_or_frgt == 'C')
   printf("Cruiser\n");

   printf("Frigate\n"); (1)
```

FIGURE 4.4 Flowcharts of if Statements with (a) Two Alternatives and (b) One Alternative


F
rest_heart_rate > 56
T
Display "Keep up your exercise program"
Display "Your heart is in excellent condition"
(a)
F
x != 0
T
product = product * x;
(b)

- **if Statement with One Alternatives:**
- You also can write if statements with a single alternative that executes only when the condition is true.
- *If statement from figure 4.4(b)*

```
/* Multiply Product by a nonzero x */
if (x != 0.0)
   product = product * x;
```

```
if (crsr_or_frgt == 'C')
    printf("Cruiser\n");
printf("Combat ship\n"); (1)
```

```
if crsr_or_frgt == 'C‘                    /* error - missing parentheses */(2)
    printf("Cruiser\n");
printf("Combat ship\n");
```

```
if (crsr_or_frgt == 'C');                 /* error - improper placement of ;*/ (3)
        printf("Cruiser\n");
printf("Combat ship\n");
```

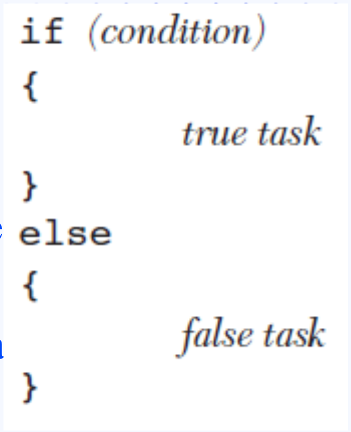
4.4 IF STATEMENTS WITH COMPOUND STATEMENTS

```
if (students_today > students_yesterday) {

        int increase = students_today - students_yesterday;
        float increase_pct = 100.0 * increase / students_yesterday;
        printf("Student attendance increased by %.2f%%\n", increase_pct);
}
```

```
  if (ctri <= MAX_SAFE_CTRI) {
    printf("Car #%d: safe\n", auto_id);
    safe = safe + 1;

    printf("Car #%d: unsafe\n", auto_id);
    unsafe = unsafe + 1;
  } (1)
```

what if we omit the braces?

PROGRAM STYLE



```
if (condition)
{
        true task
}
else
{
        false task
}
```

- ➢ We enclose a compound statement that is a true task or braces. The placement of the braces is a matter of personal pre
- ➢ Some programmers prefer to type each brace on its own line a braces:
- ➢ Some programmers prefer to use braces around all true and false tasks whether compound or not, so that all if statements in a program have a consistent style. We recommend enclosing both the true and the false tasks in braces if either is a compound statement.
- ➢ Whichever style you choose, make sure you apply it consistently.

TRACING AN IF STATEMENT

A **hand trace, or desk check** , is a careful, step-by-step simulation on paper of how the computer executes the algorithm or statement

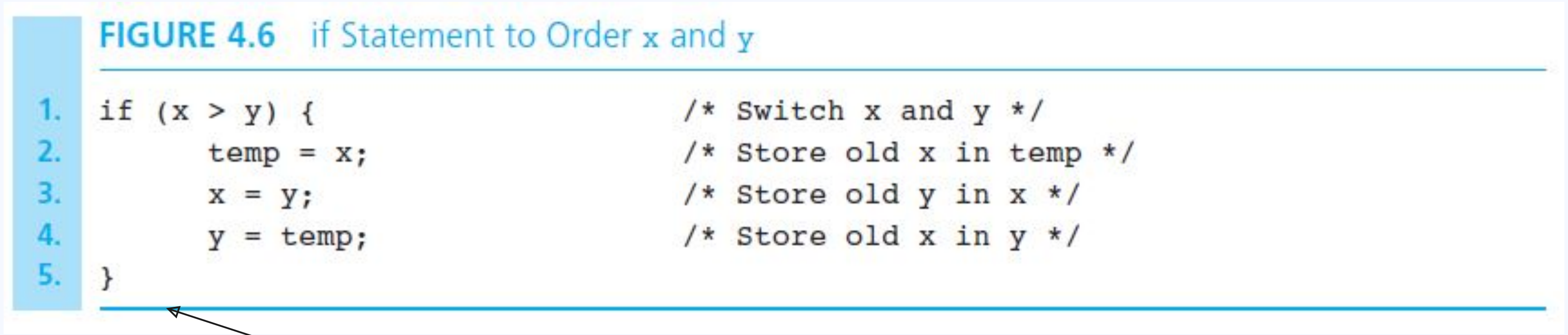


FIGURE 4.6 if Statement to Order x and y

```
if (x > y) {                    /* Switch x and y */
      temp = x;                 /* Store old x in temp */
      x = y;                    /* Store old y in x */
      y = temp;                 /* Store old x in y */
}
```

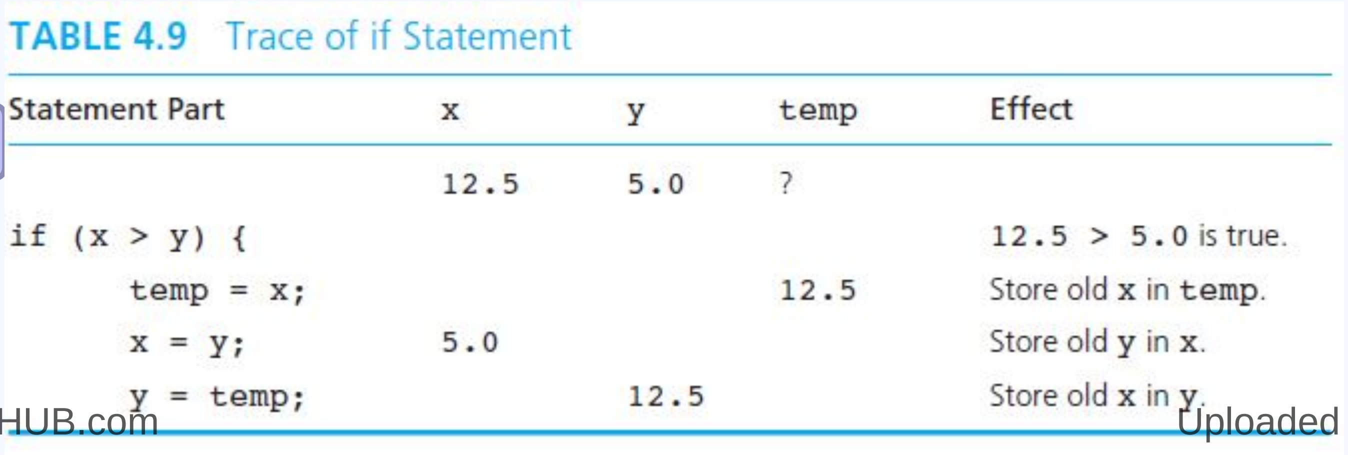


TABLE 4.9 Trace of if Statement

Statement Part	x	y	temp	Effect
	12.5	5.0	?	
`if (x > y) {`				`12.5 > 5.0` is true.
`temp = x;`			12.5	Store old x in temp.
`x = y;`	5.0			Store old y in x.
`y = temp;`		12.5		Store old x in y.

4.5 DECISION STEPS IN ALGORITHMS

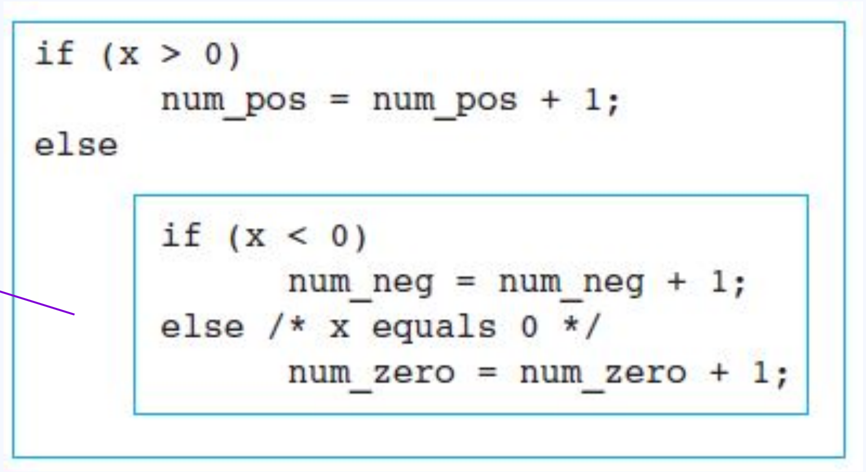
➢ **Decision Steps :**.Algorithm steps that select from a choice of actions.

CASE STUDY (Homework) P.195 - 204

Water Bill Problem

4.7 NESTED IF STATEMENTS AND MULTIPLE-ALTERNATIVE DECISIONS

False Task



```
if (x > 0)
        num_pos = num_pos + 1;
else

        if (x < 0)
                num_neg = num_neg + 1;
        else /* x equals 0 */
                num_zero = num_zero + 1;
```

Example 4.15

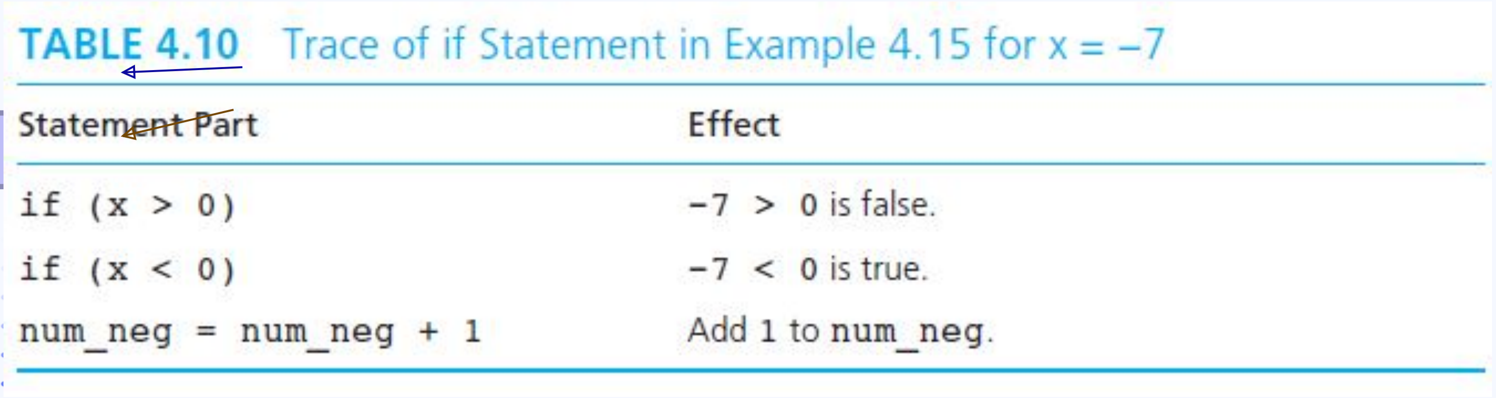


TABLE 4.10 Trace of if Statement in Example 4.15 for x = –7

Statement Part	Effect
`if (x > 0)`	`-7 > 0` is false.
`if (x < 0)`	`-7 < 0` is true.
`num_neg = num_neg + 1`	Add 1 to `num_neg`.

COMPARISON OF NESTED IF AND SEQUENCE OF IFS

- Beginning programmers sometimes prefer to use a sequence of if statements rather than a single nested if statement.

For Example: the nested if statement in **Example 4.15** is rewritten as a sequence of if statements.

```
if (x > 0)
  num_pos = num_pos + 1;
if (x < 0)
  num_neg = num_neg + 1;
if (x == 0)
  num_zero = num_zero + 1;
```

MULTIPLE-ALTERNATIVE DECISION FORM OF NESTED IF

- In situations like **Example 4.15** in which each false task (except possibly the last) is followed by an if-then-else statement, you can code the nested if as the **multiple-alternative decision**.

```
if (x > 0)
   num_pos = num_pos + 1;
else if (x < 0)
   num_neg = num_neg + 1;
else /* x equals 0 */
   num_zero = num_zero + 1;
```

ORDER OF CONDITIONS IN A MULTIPLE-ALTERNATIVE DECISION

```
/* Display perception of noise loudness */ noise_db = 62;

if (noise_db <= 50)
  printf("%d-decibel noise is quiet.\n", noise_db);
else if (noise_db <= 70)
  printf("%d-decibel noise is intrusive.\n", noise_db);
else if (noise_db <= 90)
  printf("%d-decibel noise is annoying.\n", noise_db);
else if (noise_db <= 110)
  printf("%d-decibel noise is very annoying.\n", noise_db);
else
  printf("%d-decibel noise is uncomfortable.\n", noise_db);
```

EXAMPLE 4.16

- **When more than one condition in a multiple-alternative decision is true, only the task following the first true condition executes.**(1)

```
/* incorrect perception of noise loudness */ (2)

if (noise_db <= 110)
   printf("%d-decibel noise is very annoying.\n", noise_db);
else if (noise_db <= 90)
   printf("%d-decibel noise is annoying.\n", noise_db);
else if (noise_db <= 70)
   printf("%d-decibel noise is intrusive.\n", noise_db);
else if (noise_db <= 50)
   printf("%d-decibel noise is quiet.\n", noise_db);
else
   printf("%d-decibel noise is uncomfortable.\n", noise_db);
```

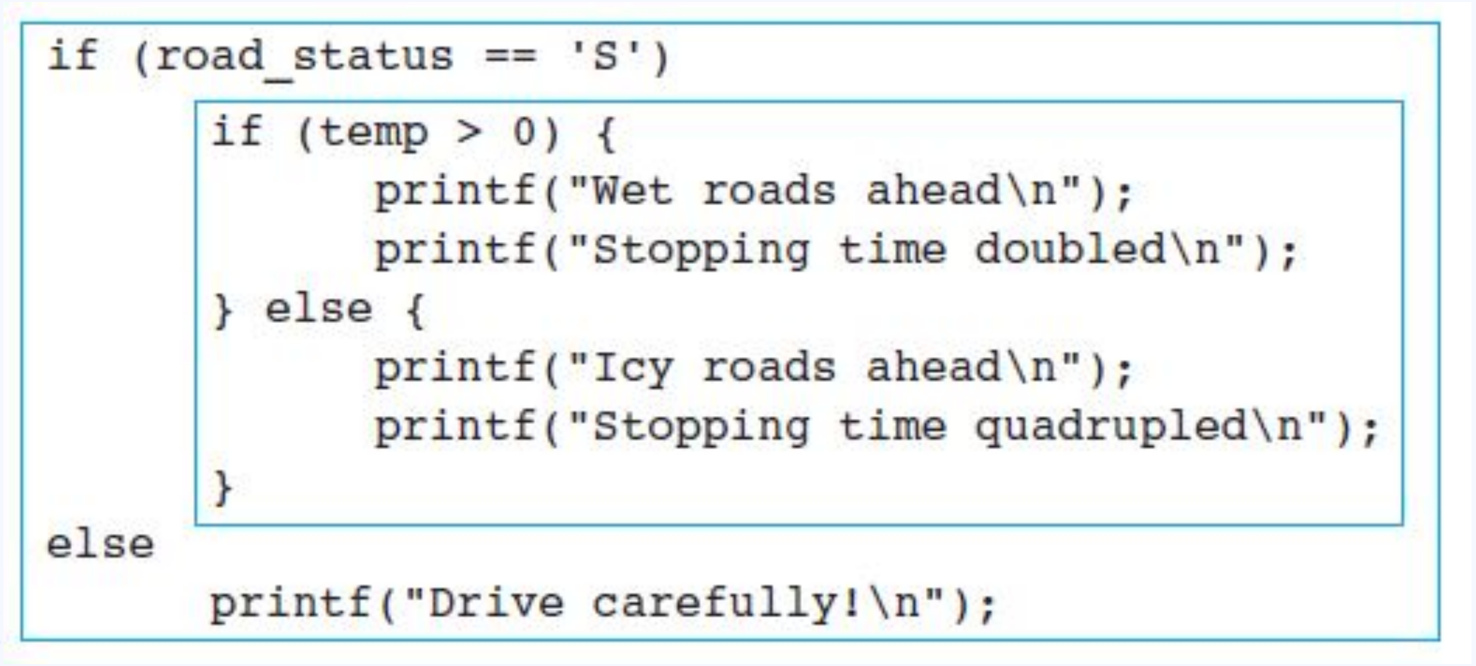
NESTED IF STATEMENTS WITH MORE THAN ONE VARIABLE

```c
/* Print a message if all criteria are met. */
if (marital_status == 'S')
     if (gender == 'M')
          if (age >= 18  &&  age <= 26)
               printf("All criteria are met.\n");
```

An equivalent statement that uses a single if with a compound condition follows

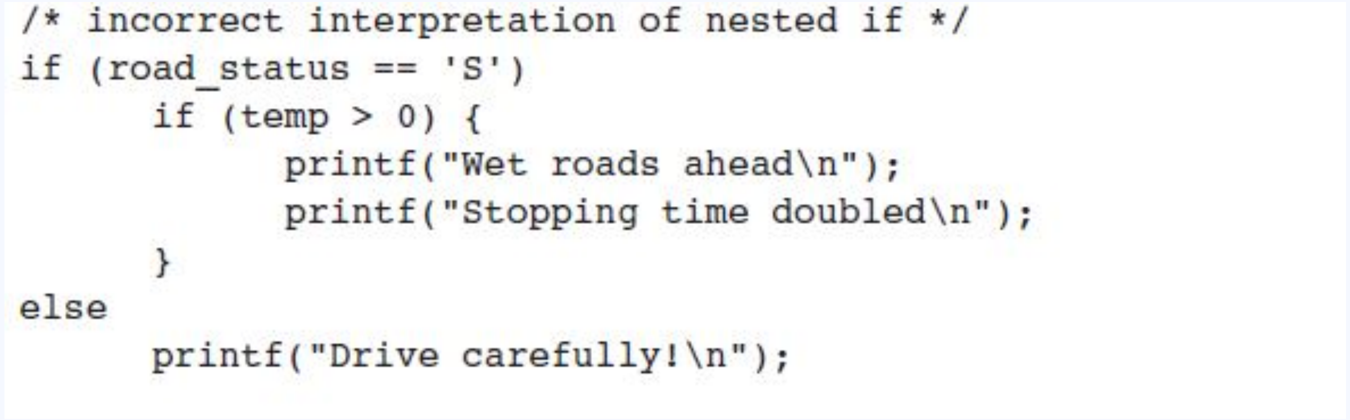
```c
if (marital_status == 'S'  &&  gender == 'M'
     &&  age >= 18  &&  age <= 26)
          printf("All criteria are met.\n");
```

- When you are writing a nested if statement, you should know that C associates an else with the most recent incomplete if



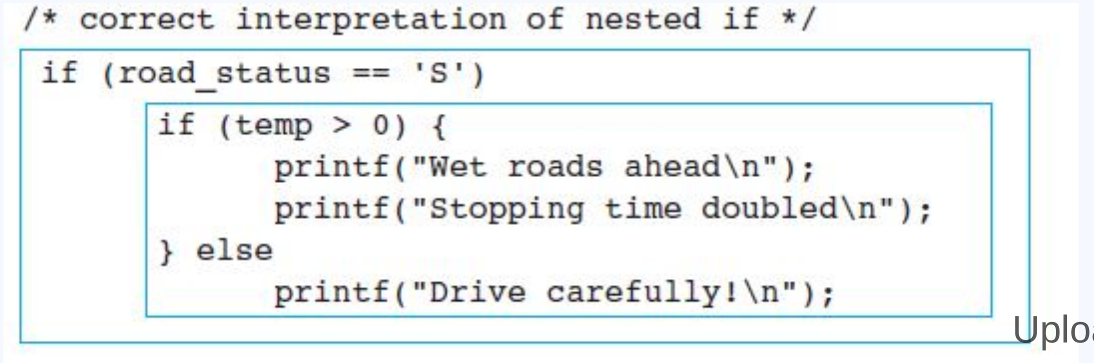
```
if (road_status == 'S')
      if (temp > 0) {
            printf("Wet roads ahead\n");
            printf("Stopping time doubled\n");
      } else {
            printf("Icy roads ahead\n");
            printf("Stopping time quadrupled\n");
      }
else
      printf("Drive carefully!\n");
```

- For example, if the first else of the road sign decision were omitted, the following would be left:



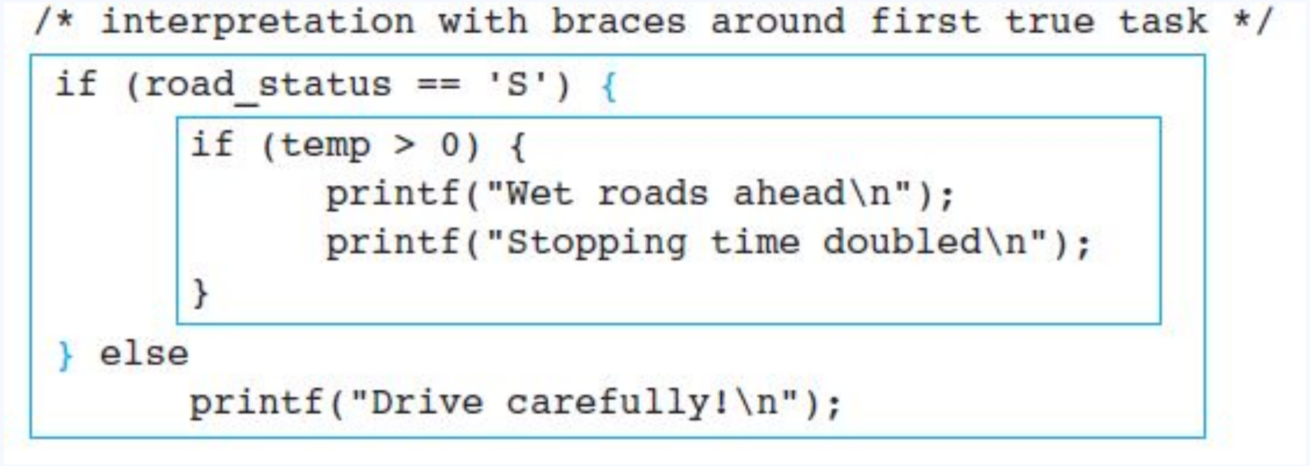
```
/* incorrect interpretation of nested if */
if (road_status == 'S')
      if (temp > 0) {
            printf("Wet roads ahead\n");
            printf("Stopping time doubled\n");
      }
else
      printf("Drive carefully!\n");
```

(1)



```
/* correct interpretation of nested if */
if (road_status == 'S')
      if (temp > 0) {
            printf("Wet roads ahead\n");
            printf("Stopping time doubled\n");
      } else
            printf("Drive carefully!\n");
```

- To force the else to be the false branch of the first if , we place braces around the true task of the first decision.



```
/* interpretation with braces around first true task */
if (road_status == 'S') {
        if (temp > 0) {
                printf("Wet roads ahead\n");
                printf("Stopping time doubled\n");
        }
} else
        printf("Drive carefully!\n");
```

- Figure 4.12 p.214

- Change the initial condition so the branches were switched, and the multiple-alternative structure would work

```
if (road_status == 'D') {
   printf("Drive carefully!\n");
} else if (temp > 0) {
   printf("Wet roads ahead\n");
   printf("Stopping time doubled\n");
} else {
   printf("Icy roads ahead\n");
   printf("Stopping time quadrupled\n");
}
```

4.8 THE SWITCH STATEMENT

- The switch statement may also be used in C to select one of several alternatives.

- The switch statement is especially useful when the selection is based on the value of a single variable or of a simple expression (called the controlling expression).

- The value of this expression may be of type int or char , but not of type double or string.

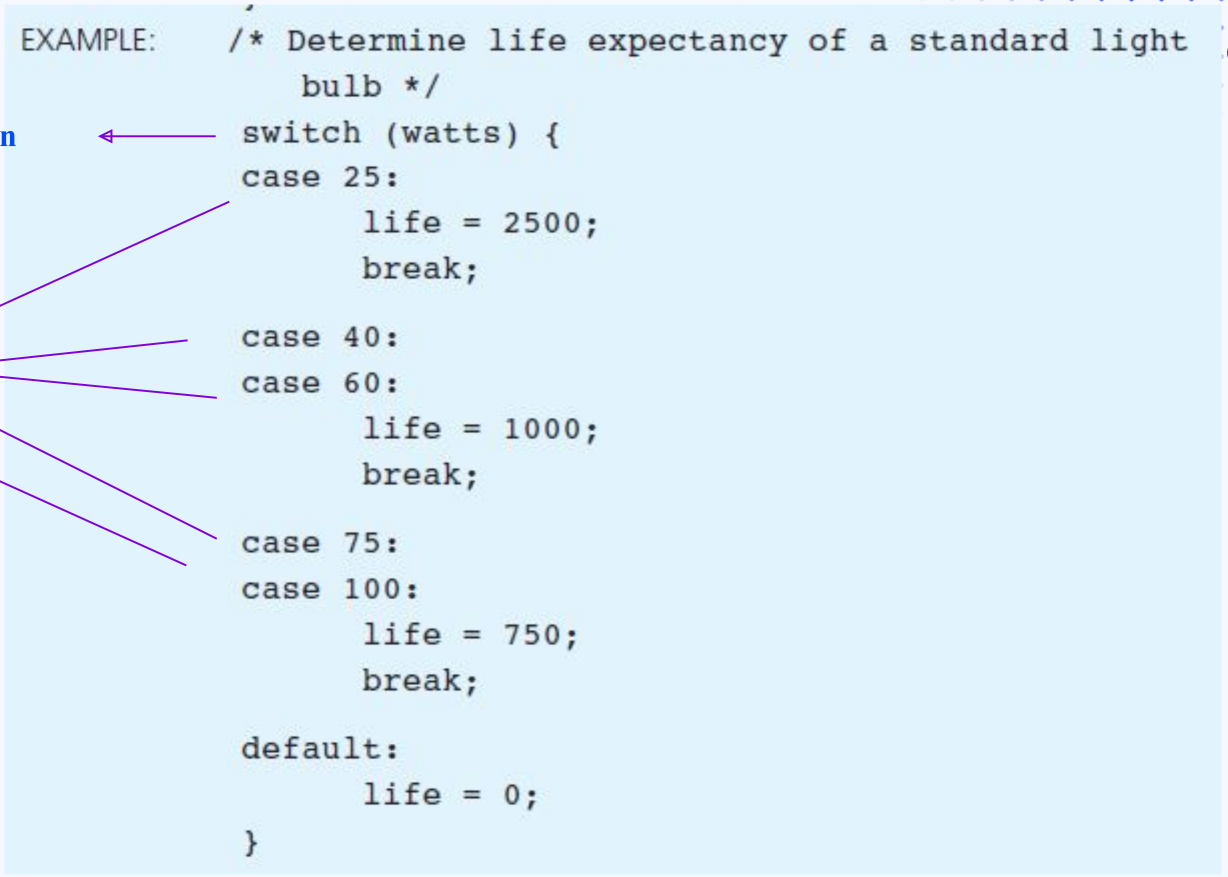
- Example 4.20 p.218

THE SWITCH STATEMENT RULES

- It is important to remember that type int and char values may be used as case labels, but strings and type double values cannot be used.

- Another common error is the omission of the break statement at the end of one alternative. In such a situation, execution "falls through" into the next alternative.

- Forgetting the closing brace of the switch statement body is also easy to do. If the brace is missing and the switch has a default label, the statements following the switch statement become part of the default case.



```
EXAMPLE:    /* Determine life expectancy of a standard light
               bulb */
             switch (watts) {
             case 25:
                      life = 2500;
                      break;

             case 40:
             case 60:
                      life = 1000;
                      break;

             case 75:
             case 100:
                      life = 750;
                      break;

             default:
                      life = 0;
             }
```

The controlling expression

Label set

THE SWITCH STATEMENT RULES

- You can use a **nested if statement**, which is more general than the switch statement, to implement any multiple-alternative decision.
- The switch as described in the syntax display is more readable in many contexts and should be used whenever practical.
- Case labels that contain type **double** values or **strings** are not permitted.
- You should use the switch statement when each label set contains a reasonable number of case labels (a maximum of ten). (1)
- You should include a default label in switch statements wherever possible. (2)

4.9 COMMON PROGRAMMING ERRORS

- Displays Condition is true for all values of x .

- /*consider x is 5 */ (1)

```
if (0 <= x <= 4)
```

=> Compiler can’t detect!

```
printf("Condition is true\n");
```

We should use:

```
(0 <= x && x <= 4)
```

What do we call such an error?

4.9 COMMON PROGRAMMING ERRORS

- Remember that the C equality operator is == not

```
if (x = 10)              => Compiler can't detect! (1)
        printf("x is 10");
```

always prints x is 10 , regardless of the value of x .

4.9 COMMON PROGRAMMING ERRORS

- parenthesize the condition of an if statement.
- enclose in braces a single-alternative if used as a true task within a double-alternative if. (1)

/*Correct Version*/

```
if (x > y)
    if (x % 2 == 0)
        printf("x is even\n");

  else
      printf("x is smaller than y\n");
```

```
if (x > y) {
    if (x % 2 == 0) {
    printf("x is even\n");

    }
  } else {
      printf("x is greater than
y\n");
  }
```

- If x = 5 , y = 4 => “x is smaller than y”

4.9 COMMON PROGRAMMING ERRORS

- **Enclose in braces a compound statement used as a true task or false task.** (1)

```
if (x > 0)
    sum = sum + x;
    printf("Greater than zero\n");
else
    printf("Less than or equal to zero\n");
```

4.9 COMMON PROGRAMMING ERRORS

- When writing a nested if statement, try to select the conditions so that you can use the multiple-alternative format.
- When possible, the logic should be constructed so each intermediate condition falls on the false branch of the previous decision.
- If more than one condition can be true at the same time, place the most restrictive condition first

Refernces

Problem Solving and Program Design in C, 7th Ed., by Jeri R. Hanly and Elliot B. Koffman