Chapter 4: Selection Structures: if and switch Statements

Problem Solving & Program Design in C

Eighth Edition

By Jeri R. Hanly &
Elliot B. Koffman

Addison Wesley
is an imprint of

PEARSON

Outline

4.1 Control Structure

- Control structure
  - A combination of individual instructions into a single logical unit with one entry point and one exit point
  - Control the flow of execution in a program
- Three kinds of control structures
  - sequential
  - selection
  - repetition

Control Structure (cont.)

- Compound statement
  - a group of statements bracketed by { and } that are executed sequentially
  - a function body consists of a single compound statement
- Selection control structure
  - Chooses among alternative program statements

4.2 Conditions

- Condition
  - An expression that is either false(0) or true(1)
- Condition establishes a criterion for either executing or skipping a group of statements.

4.2.1 Relational and equality operators

- One of the following forms
  - Variable relational-operator Variable
  - Variable relational-operator Constant
  - Variable equality-operator Variable
  - Variable equality-operator Constant

Example 4.1

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

4.2.2 Logical Operators

- logical expression
  - An expression that uses one or more of the logical operators && (and), || (or), ! (not)
- logical complement (negation)
  - The complement of a condition has the value 1(true) when the condition's value is 0(false) and vice versa
- Table 4.3 & Table 4.4 & Table 4.5 (P.197)

Table 4.6 Operator Precedence

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

Example 4.2

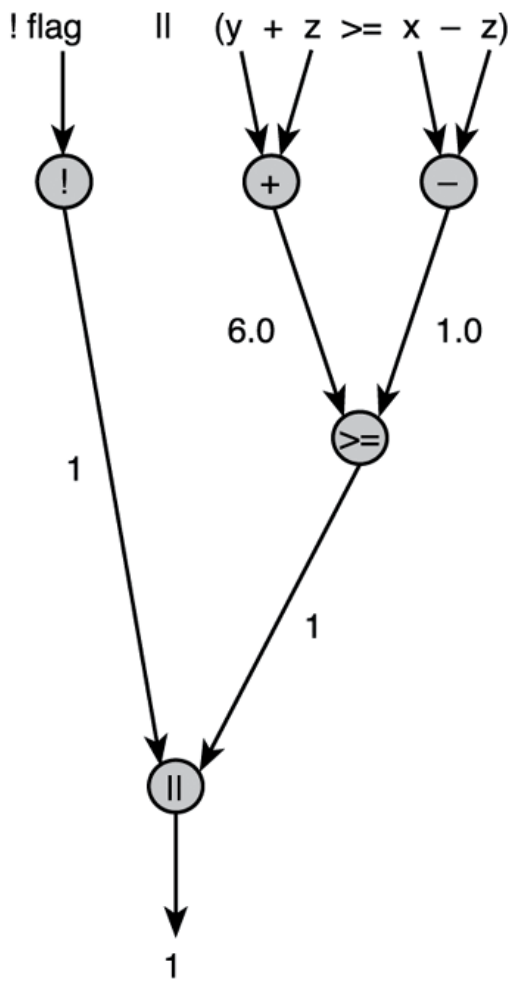
- Expression 1 to 4 below contain different operands and operators.
- Each expression's value is given in the corresponding comment, assuming x, y, and z are type double, flag is type int and the variables have the values

Example 4.2 (cont.)

- 1. ! flag → true
- 2. x + y / z <= 3.5 → false
- 3. ! flag || (y + z >= x - z) → true
  - **(Figure 4.1)**
- 4. !(flag || (y + z >= x - z)) → false

x	y	z	flag
3.0	4.0	2.0	0

Figure 4.1 Evaluation Tree and Step-by-Step Evaluation for !flag II (y + z >= x - z)



flag	y	z	x
0	4.0	2.0	3.0

! flag	||	(y + z	>=	x – z
0		4.0 2.0		3.0 2.0
1		6.0		1.0
		1		

1

Logic output

4.2.4 Short-Circuit Evaluation

- Stopping evaluation of a logical expression as soon as its value can be determined.

- ! flag || (y + z >= x - z) → true

x	y	z	flag
3.0	4.0	2.0	0

4.2.5 Writing English Conditions in C

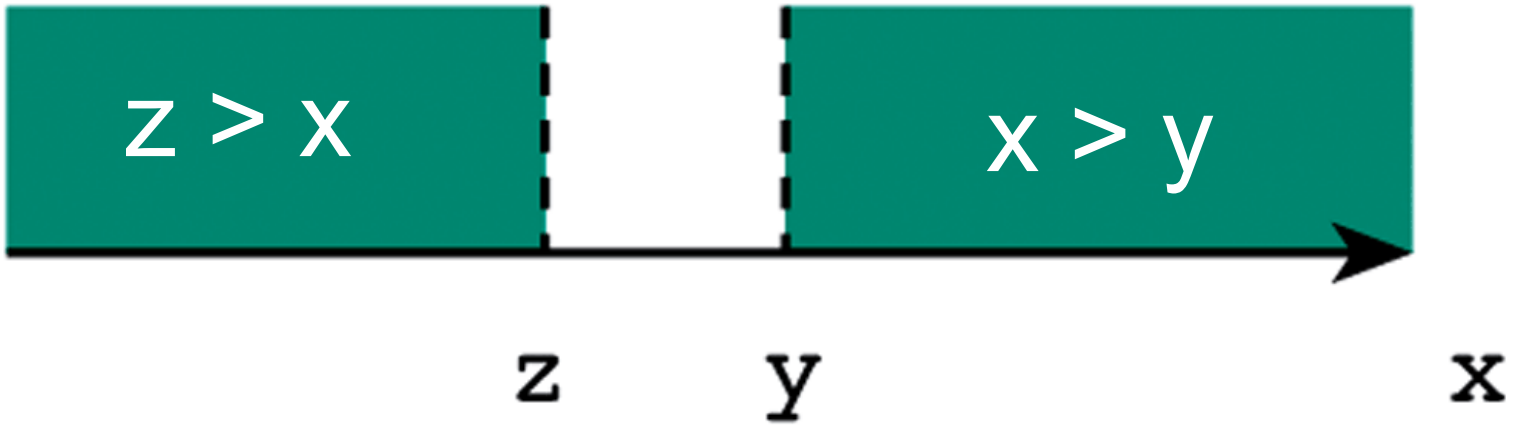
Figure 4.2 Range of True Values for min <= x && x <= max

English Conditions as C Expressions

Example 4.3 x = 3.0 y = 4.0 z = 2.0

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) z>x || x>y	!(1 && 1) is 0 (false) 0 || 0 is 0 (false)

Figure 4.3 Range of True Values for z > x || x > y



4.2.6 Comparing Characters

- 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

4.2.7 Logical Assignment

- Use assignment statements to set variables to true (nonzero) or false (0)
- Ex:

```
int age, senoir_citizen;
scanf("%d", &age);
senior_citizen = (age >= 65);
```

4.2.8 Complementing a Condition

- DeMorgan's Theorem
  - The complement of $expr_1$ && $expr_2$
    - → $comp_1$ || $comp_2$
  - The complement of $expr_1$ || $expr_2$
    - → $comp_1$ && $comp_2$
- Example (Example 4.8)

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

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

4.3 The If Statement

4.3.1 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");
```

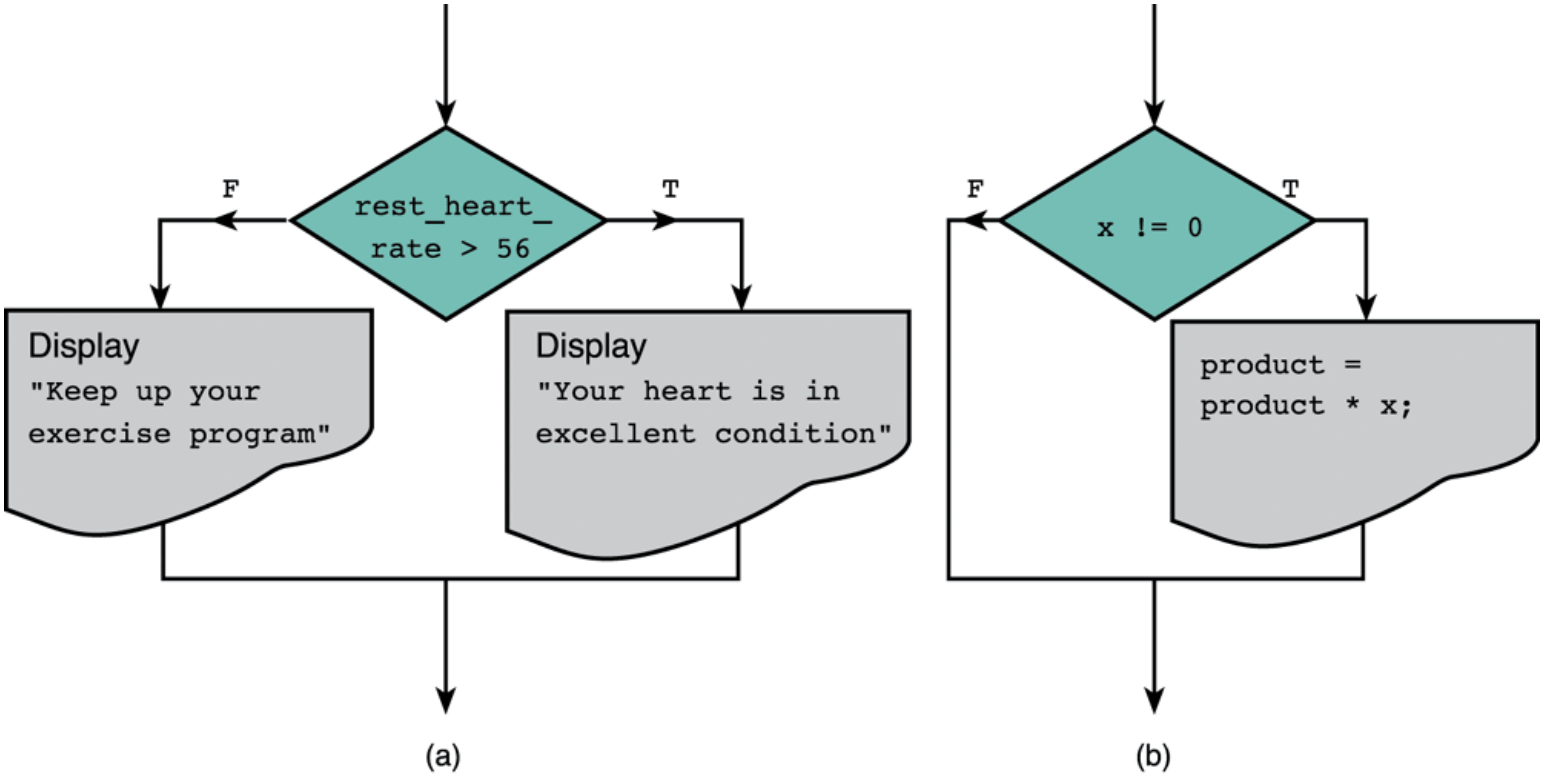
4.3.2 If statement with one alternative

```
if (x!=0.0)
        product = product * x;
```

4.3 (cont) The If Statement

- Flowchart (Figure 4.4)
  - a diagram that shows the step-by-step execution of a control structure
- Diamond-shaped box
  - represent a decision in the flowchart
- Rectangular box
  - represent an assignment statement or process

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



4.3.3 A Comparison of One and Two Alternative if Statements

- if statement (One Alternative)

FORM: **if** (***condition***)
***statement*$_T$;**

Example: if (x > 0.0)
pos_prod = pos_prod * x ;

A Comparison of One and Two Alternative if Statements (cont)

- if statement (Two Alternative)

FORM: **if** (*condition*)
statement$_T$;
else
statement$_F$;

Example:
```
if ( x >= 0.0)
    printf("positive\n");
else
    printf("negative\n");
```

4.4 if Statements with Compound Statements

4.4.1 Writing if statements with compound true or false statements

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

4.4.2 Tracing an if Statement

- hand trace (desk trace)
  - step-by-step simulation of an algorithm's execution
- Figure 4.5

Figure 4.6
if Statement to Order x and y

```
1.  if (x > y) {                    /* Switch x and y */
2.          temp = x;               /* Store old x in temp */
3.          x = y;                  /* Store old y in x */
4.          y = temp;               /* Store old x in y */
5.  }
```

Table 4.9 Tracing an 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 y in x.
y = temp ;		12.5		Store old x in y.

4.5 Decision Steps in Algorithms

- Decision step
  - an algorithm step the selects one of several actions

- *Case Study：Water Bill Problem*

Case Study ：Water Bill Problems

<Step 1> Problem

- Write a program that computes a customer's water bill.
- The bill includes a $35 water demand charge plus a consumption (use) charge of $1.10 for every thousand gallons used.
- Consumption is figured from meter readings (in thousands of gallons) taken recently and at the end of the previous quarter. If the customer's unpaid balance is greater than zero, a $2 late charge is assessed as well.

Case Study ：Water Bill Problems (cont)

<Step 2> Analysis

DATA REQUIREMENTS

– Problem Constants

- DEMAND_CHG 35.00
- PRE_1000_CHG 1.10
- LATE_CHG 2.00

– Problem Inputs

- int previous
- int current
- double unpaid

Case Study ：Water Bill Problems (cont)

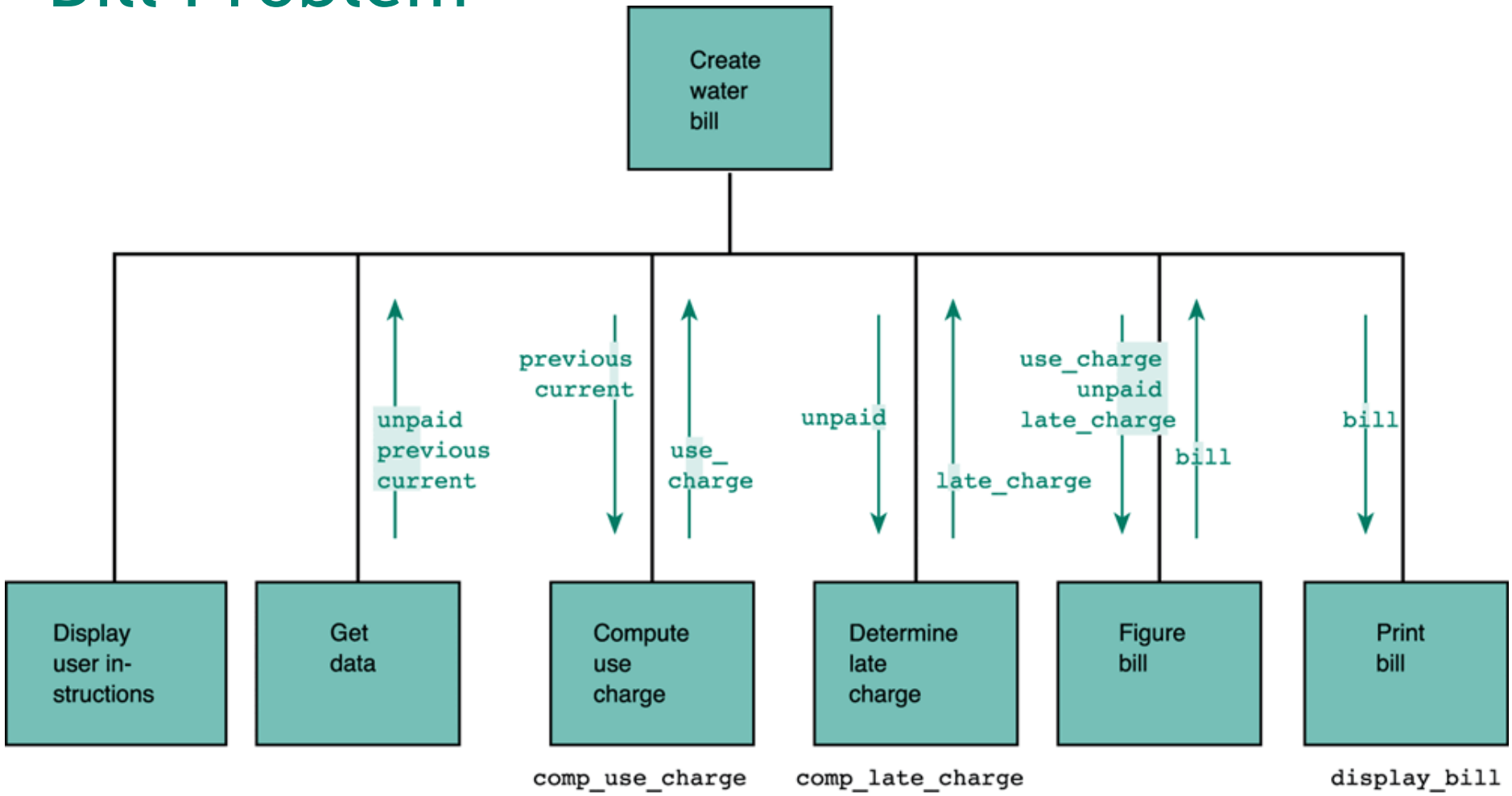
<Step 2> Analysis (cont)

- Problem Outputs
  - double bill
  - double use_charge
  - double late_charge
- Relevant Formulas
  - *water bill = demand charge + use charge + unpaid balance + applicable late charge*

Case Study ：Water Bill Problems (cont)

- Design
  - Initial algorithm
    1. Display user instructions.
    2. Get data:
       - unpaid balance, previous and current meter readings.
    3. Compute use charge.
    4. Determine applicable late charge.
    5. Figure bill amount.
    6. Display the bill amount and charges.

Figure 4.7 Structure Chart for Water Bill Problem



Case Study : Water Bill Problems (cont)

Analysis and Design of COMP_USE_CHARGE

- Input parameters
  - int previous
  - int current
- Return value
  - double use_charge
- Program variable
  - int used
- Relevant formulas
  - *used = current meter reading – previous meter reading*
  - *use charge = used x charge per thousand gallons*

Case Study ：Water Bill Problems (cont)

Analysis and Design of COMP_USE_CHARGE

- Algorithm for COMP_USE_CHARGE

1. used = current – previous
2. use charge = used * charge per thousand gallons

Case Study : Water Bill Problems (cont)

Analysis and Design of COMP_LATE_CHARGE

- Input parameter
  - double unpaid
- Return value
  - double late_charge
- Algorithm for COMP_LATE_CHARGE

```
1. if unpaid > 0
     assess late charge
   else
     assess no late charge
```

***Pseudocode*:**
a combination of English and C constructs to describe algorithm steps

Case Study : Water Bill Problems (cont)

Analysis and Design of DISPLAY_BILL

- Input parameters
  - double late_charge
  - double bill
  - double unpaid
- Algorithm for DISPLAY_BILL
  - 1. if late_charge > 0

    display late charge and unpaid balance
  - 2. Display the bill amount.

Case Study Water Bill Problems (cont)

- Implementation (Figure 4.8)
- Testing (Figure 4.9)

Figure 4.8 Program for Water Bill Problem

```
1.  /*
2.   * Computes and prints a water bill given an unpaid balance and previous and
3.   * current meter readings. Bill includes a demand charge of $35.00, a use
4.   * charge of $1.10 per thousand gallons, and a surcharge of $2.00 if there is
5.   * an unpaid balance.
6.   */
7.
8.  #include <stdio.h>
9.
10. #define DEMAND_CHG  35.00  /* basic water demand charge                    */
11. #define PER_1000_CHG 1.10  /* charge per thousand gallons used             */
12. #define LATE_CHG     2.00  /* surcharge assessed on unpaid balance         */
13.
14. /* Function prototypes                                                     */
15. void instruct_water(void);
16.
17. double comp_use_charge(int previous, int current);
18.
19. double comp_late_charge(double unpaid);
20.
21. void display_bill(double late_charge, double bill, double unpaid);
22.
23. int
24. main(void)
25. {
26.         int    previous;    /* input - meter reading from previous quarter
27.                                in thousands of gallons                     */
28.         int    current;     /* input - meter reading from current quarter  */
29.         double unpaid;      /* input - unpaid balance of previous bill     */
30.         double bill;        /* output - water bill                         */
31.         int    used;        /* thousands of gallons used this quarter      */
32.         double use_charge;  /* charge for actual water use                 */
33.         double late_charge; /* charge for nonpayment of part of previous
34.                                balance                                     */
35.
36.         /* Display user instructions.                                      */
37.         instruct_water();
38.
39.         /* Get data: unpaid balance, previous and current meter
40.            readings.                                                       */
```

(continued)

Figure 4.8 Program for Water Bill Problem (cont'd)

```
41.         printf("Enter unpaid balance> $");
42.         scanf("%lf", &unpaid);
43.         printf("Enter previous meter reading> ");
44.         scanf("%d", &previous);
45.         printf("Enter current meter reading> ");
46.         scanf("%d", &current);
47.
48.         /* Compute use charge.                                          */
49.         use_charge = comp_use_charge(previous, current);
50.
51.         /* Determine applicable late charge                             */
52.         late_charge = comp_late_charge(unpaid);
53.
54.         /* Figure bill.                                                 */
55.         bill = DEMAND_CHG + use_charge + unpaid + late_charge;
56.
57.         /* Print bill.                                                  */
58.         display_bill(late_charge, bill, unpaid);
59.
60.         return (0);
61. }
62.
63. /*
64.  * Displays user instructions
65.  */
66. void
67. instruct_water(void)
68. {
69.       printf("This program figures a water bill ");
70.       printf("based on the demand charge\n");
71.       printf("($%.2f) and a $%.2f per 1000 ", DEMAND_CHG, PER_1000_CHG);
72.       printf("gallons use charge.\n\n");
73.       printf("A $%.2f surcharge is added to ", LATE_CHG);
74.       printf("accounts with an unpaid balance.\n");
75.       printf("\nEnter unpaid balance, previous ");
76.       printf("and current meter readings\n");
77.       printf("on separate lines after the prompts.\n");
78.       printf("Press <return> or <enter> after ");
79.       printf("typing each number.\n\n");
80. }
81.
```

(continued)

Figure 4.8 Program for Water Bill Problem (cont'd)

```
82.  /*
83.   * Computes use charge
84.   * Pre: previous and current are defined.
85.   */
86.  double
87.  comp_use_charge(int previous, int current)
88.  {
89.        int used;  /* gallons of water used (in thousands)                    */
90.        double use_charge;  /* charge for actual water use                    */
91.
92.        used = current - previous;
93.        use_charge = used * PER_1000_CHG;
94.
95.        return (use_charge);
96.  }
97.
98.  /*
99.   * Computes late charge.
100.  * Pre : unpaid is defined.
101.  */
102. double
103. comp_late_charge(double unpaid)
104. {
105.       double late_charge; /* charge for nonpayment of part of previous balance  */
106.
107.       if (unpaid > 0)
108.             late_charge = LATE_CHG; /* Assess late charge on unpaid balance.    */
109.       else
110.             late_charge = 0.0;
111.
112.       return (late_charge);
113. }
114.
115. /*
116.  * Displays late charge if any and bill.
117.  * Pre : late_charge, bill, and unpaid are defined.
118.  */
119. void
120. display_bill(double late_charge, double bill, double unpaid)
```

(continued)

Figure 4.8 Program for Water Bill Problem (cont'd)

```
121. {
122.       if (late_charge > 0.0) {
123.             printf("\nBill includes $%.2f late charge", late_charge);
124.             printf(" on unpaid balance of $%.2f\n", unpaid);
125.       }
126.       printf("\nTotal due = $%.2f\n", bill);
127. }
```

Figure 4.9 Sample Run of Water Bill Program

```
This program figures a water bill based on the demand charge
($35.00) and a $1.10 per 1000 gallons use charge.

A $2.00 surcharge is added to accounts with an unpaid balance.

Enter unpaid balance, previous and current meter readings
on separate lines after the prompts.
Press <return> or <enter> after typing each number.

Enter unpaid balance> $71.50
Enter previous meter reading> 4198
Enter current meter reading> 4238

Bill includes $2.00 late charge on unpaid balance of $71.50

Total due = $152.50
```

Program Style

- Consistent use of names in functions
- Cohesive functions
  - a function that performs a single operation
- Using constant macros to enhance readability and ease maintenance

4.6 More Problem Solving

- Output of the step
  - a step gives a new value to a variable
  - the variable is considered an output to the step
- Input of the step
  - a step displays a variable's value or uses a variable in a computation without changing its value
- *A variable can be an output of the step and also an input of another step*

Case Study：Water Bill with Conservation Requirement (cont)

<Step 1> Problem

- We need to modify the water bill program so that customers who fail to meet conservation requirements are charged for all their water use at twice the rate of customers who meet the guidelines.
- Residents of this water district are required to use no more than 95% of the amount of water they used in the same quarter last year in order to qualify for the lower use rate of $1.10 per thousand gallons.

Case Study：Water Bill with Conservation Requirement (cont)

<Step 2> Analysis (Additions to data requirements)

- Problem constants
  - OVER_CHG_RATE 2.0
  - CONSERV_RATE 95
- Problem inputs
  - int use_last_year

Case Study : Water Bill with Conservation Requirement (cont)

<Step 3> Algorithm for COMP_USE_CHANGE

- 1. used is current – previous
- 2. if guidelines are met

```
        use_charge = used * PER_1000_CHANGE
    else
        notify customer of overuse
        use_charge = used * overuse_chg_rate *
                                    PER_1000_CHANGE
```

(Figure 4.10)

Figure 4.10 Function comp_use_charge Revised

```
1.  /*
2.   * Computes use charge with conservation requirements
3.   * Pre: previous, current, and use_last_year are defined.
4.   */
5.  double
6.  comp_use_charge(int previous, int current, int use_last_year)
7.  {
8.        int used;  /* gallons of water used (in thousands)                */
9.        double use_charge;  /* charge for actual water use                */
```

(continued)

Figure 4.10 Function comp_use_charge Revised (cont'd)

```
10.       used = current - previous;
11.       if (used <= CONSERV_RATE / 100.0 * use_last_year) {
12.             /* conservation guidelines met */
13.             use_charge = used * PER_1000_CHG;
14.       } else {
15.             printf("Use charge is at %.2f times ", OVERUSE_CHG_RATE);
16.             printf("normal rate since use of\n");
17.             printf("%d units exceeds %d percent ", used, CONSERV_RATE);
18.             printf("of last year's %d-unit use.\n", use_last_year);
19.             use_charge = used * OVERUSE_CHG_RATE * PER_1000_CHG;
20.       }
21.
22.       return (use_charge);
23. }
```

4.7 Nested if Statements and Multiple-Alternative Decisions

- nested if statement
  - an if statement with another if statement as its true task or its false task
- Example 4.15

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

4.7.2 Multiple-Alternative Decision Form of Nested if

Syntax:

if (*condition*$_1$)
    statement$_1$
else if (*condition*$_2$)
    statement$_2$
    .
    .
else if (*condition*$_n$)
    statement$_n$
else
    statement$_e$

Example:

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

Example 4.17

- Use a multiple-alternative if statement to implement a decision table that describes several alternatives. (Figure 4.11)

Salary Range($)	Base Tax($)	Percentage of Excess
0.00- 1,4999.99	0.00	15
15,000.00- 29,999.99	2,250.00	18
30,000.00- 49,999.99	5,400.00	22
50,000.00- 79,999.99	11,000.00	27
80,000.00- 150,000.00	21,600.00	33

Figure 4.11 Function comp_tax

```
1.  /*
2.   * Computes the tax due based on a tax table.
3.   * Pre : salary is defined.
4.   * Post: Returns the tax due for 0.0 <= salary <= 150,000.00;
5.   *       returns -1.0 if salary is outside the table range.
6.   */
7.  double
8.  comp_tax(double salary)
9.  {
10.    double tax;
11.
12.    if (salary < 0.0)
13.         tax = -1.0;
```

(continued)

Figure 4.11 Function comp_tax (cont'd)

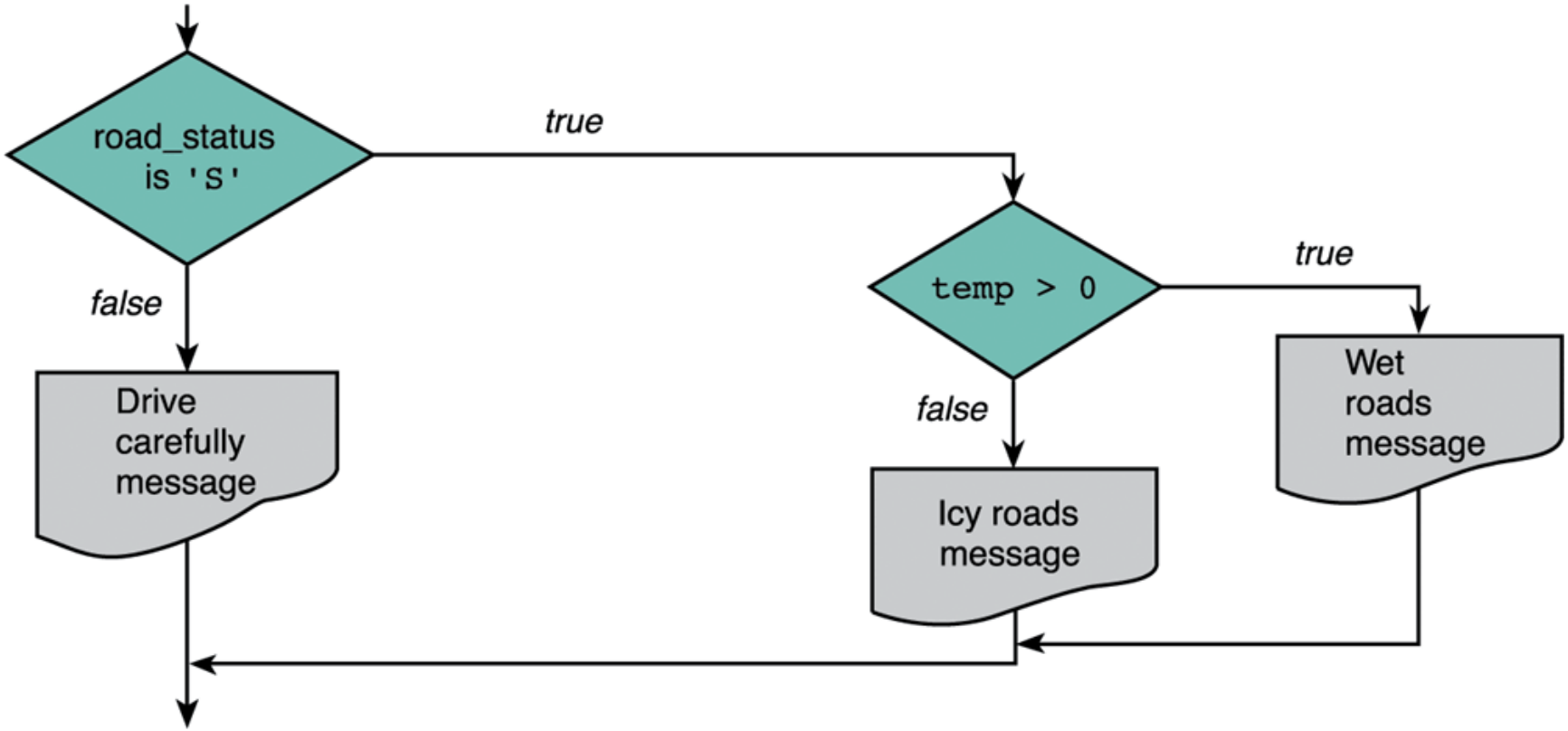
```
14.     else if (salary < 15000.00)                    /* first range      */
15.          tax = 0.15 * salary;
16.     else if (salary < 30000.00)                    /* second range     */
17.          tax = (salary - 15000.00) * 0.18 + 2250.00;
18.     else if (salary < 50000.00)                    /* third range      */
19.          tax = (salary - 30000.00) * 0.22 + 5400.00;
20.     else if (salary < 80000.00)                    /* fourth range     */
21.          tax = (salary - 50000.00) * 0.27 + 11000.00;
22.     else if (salary <= 150000.00)                  /* fifth range      */
23.          tax = (salary - 80000.00) * 0.33 + 21600.00;
24.     else
25.          tax = -1.0;
26.
27.     return (tax);
28. }
```

- Use Table 4.12 to trace this function

Example 4.19 Warning signal controller (Figure 4.12)

- Control the warning signs at the exists of major tunnels.
- If roads are slick, you want to advise drivers that stopping times are doubled or quadrupled, depending on whether the roads are wet or icy.
- Access to current temperature for checking whether the temperature is below or above freezing.

Figure 4.12 Flowchart of Road Sign Decision Process



Example 4.19 (cont)

```
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 quardrupled\n");
        }
else
        printf("Drive carefully!\n");
```

Multiple-alternative decision statement for EX.4.19

```
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 quardrupled\n");
}
```

4.8 The switch Statement

- The switch statement selection is based on the value of a single variable or of a simple expression.
- Expression may be of type int or char, but not of type double or string.

Example 4.20 (Figure 4.13)

Class ID	Ship Class
B or b	Battleship
C or c	Cruiser
D or d	Destroyer
F or f	Frigate

Figure 4.13 Example of a switch Statement with Type char Case Labels

```
1.  switch (class) {
2.  case 'B':
3.  case 'b':
4.          printf("Battleship\n");
5.          break;
6.
7.  case 'C':
8.  case 'c':
9.          printf("Cruiser\n");
10.         break;
11.
12. case 'D':
13. case 'd':
14.         printf("Destroyer\n");
15.         break;
16.
17. case 'F':
18. case 'f':
19.         printf("Frigate\n");
20.         break;
21.
22. default:
23.        printf("Unknown ship class %c\n", class);
24. }
```

The switch Statement Syntax

Syntax:

```
switch (controlling expression) {
label set1
        statements1
        break;
label set2
        statements2
        break;
          …
label setn
        statementsn
        break;
default:
        statementsd
}
```

Example:

```
Switch (watts){
case 25:
        life = 2500;
        break;
case 40:
case 60:
        life = 1000;
        break;
case 75:
case100:
        life = 750;
        break;
default:
        life = 0;
}
```

Comparison of Nested if Statements and The switch Statement

- Nested if statement
  - more general to implement any multiple-alternative decision
- switch statement
  - syntax display is more readable in many contexts
  - each label set contains a reasonable number of case labels (maximum of ten)
  - default label will help you to consider what will happen if the value falls outside your set of case label values

4.9 Common Programming Errors (if Statement)

- if (0 <= x <= 4) → → → if (0<=X && x<=4)
- = → → → ==

  ex:
  ```
  If (x = 10)
        printf(" x is 10');
  ```
  – compiler can detect this error only if the first operand is not a variable
- Don't forget to enclose in braces a single-alternative if used as a true task within a double-alternative if

Example 4.19 (cont)

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

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

Common Programming Errors (switch Statement)

- Make sure the controlling expression and case labels are of the same permitted type (int or char)
- Include a default case
- Enclosed in one set of braces
- Each alternative is ended by a break statement

Chapter Review (1)

- Use control structures to control the flow of statement execution in a program.
- The compound statement is a control structure for sequential execution.
- Use selection control structures to represent decisions in a algorithm and use pseudocode to write them in algorithm.

Chapter Review (2)

- Use the relational and equality operators to compare variables and constants.
- Use the logical operators to form more complex conditions.
- Data flow information in a structure chart indicates whether a variable processed by a subproblem is used as an input or output , or as both.

Chapter Review (3)

- Nested if statements are used to represent decisions with multiple alternatives.
- The switch statement implements decisions with several alternatives, where the alternatives selected depends on the value of a variable or expression (the *controlling expression*).
- The controlling expression can be type int or char, but not type double.