Computer Programming

Selection Structures:
if and switch Statements

Control Structures

- Controls the flow of program execution
  - Sequence
  - Selection
  - Repetition
- We used sequence flow
  - Control flows from one statement to next one
  - A compound statement in braces
    - Ex: function body
- We will learn selection control statements
  - if
  - switch
- They select one statement block and executes them

Conditions

- We need conditions in selection structures

- Ex: Testing the value of a variable

  rest_heart_reate > 75

  – true (1): if greater than 75

  – false (0): otherwise

  variable relational-operator constant

  variable equality-operator constant

- C accepts any nonzero value as a true

Relational and Equality Operators

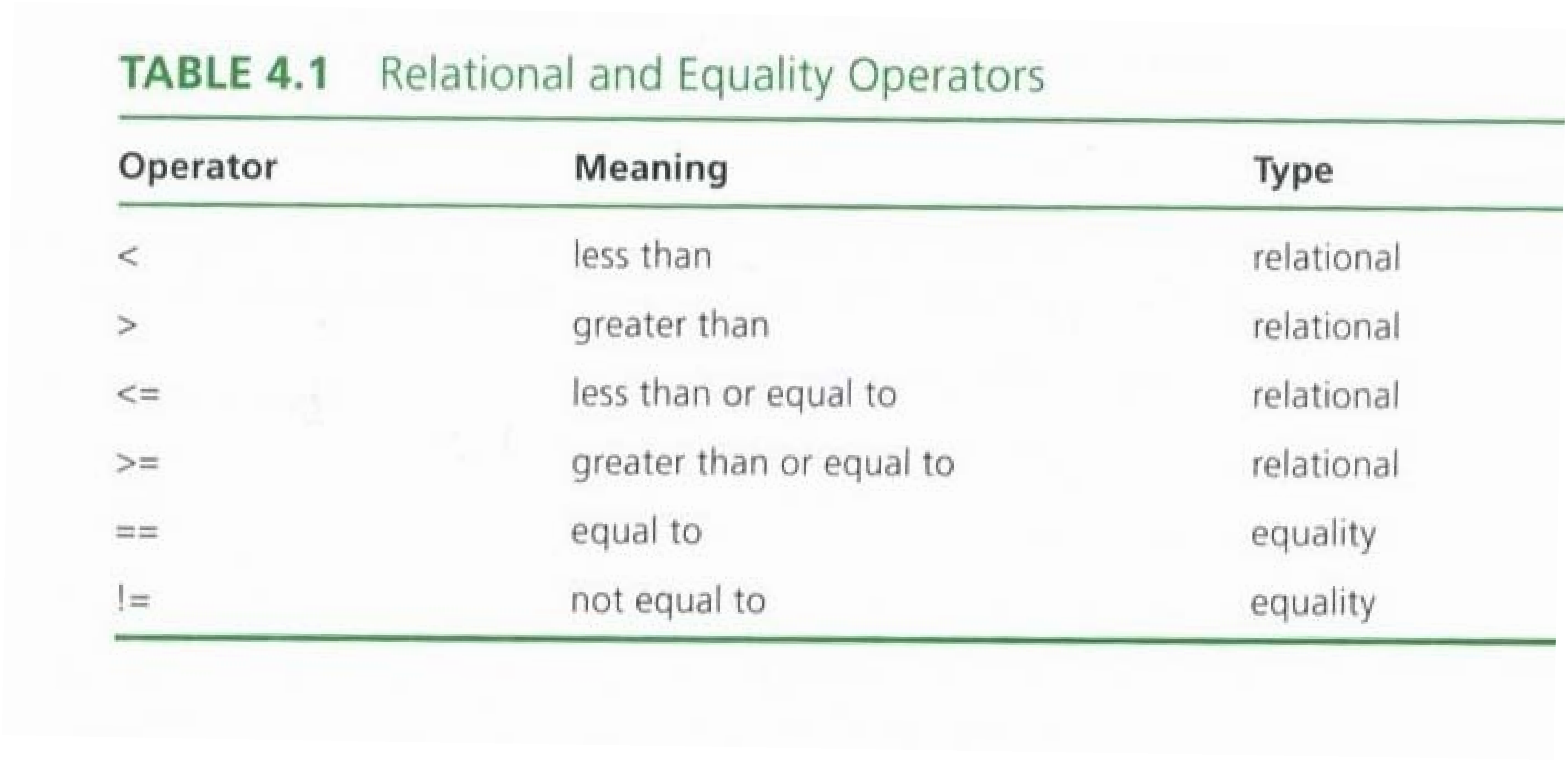


TABLE 4.1 Relational and Equality Operators

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

Logical Operators

- Used to form more complicated logical expressions
  - And (&&)
  - Or (||)
  - Not (!)
- Ex:

```
salary < MIN_SALARY || dependents > 5
temperature > 90.0 && humidity > 0.90
n >= 0 && n <= 100
!(n >= 0 && n <= 100)
```

Operator Precedence

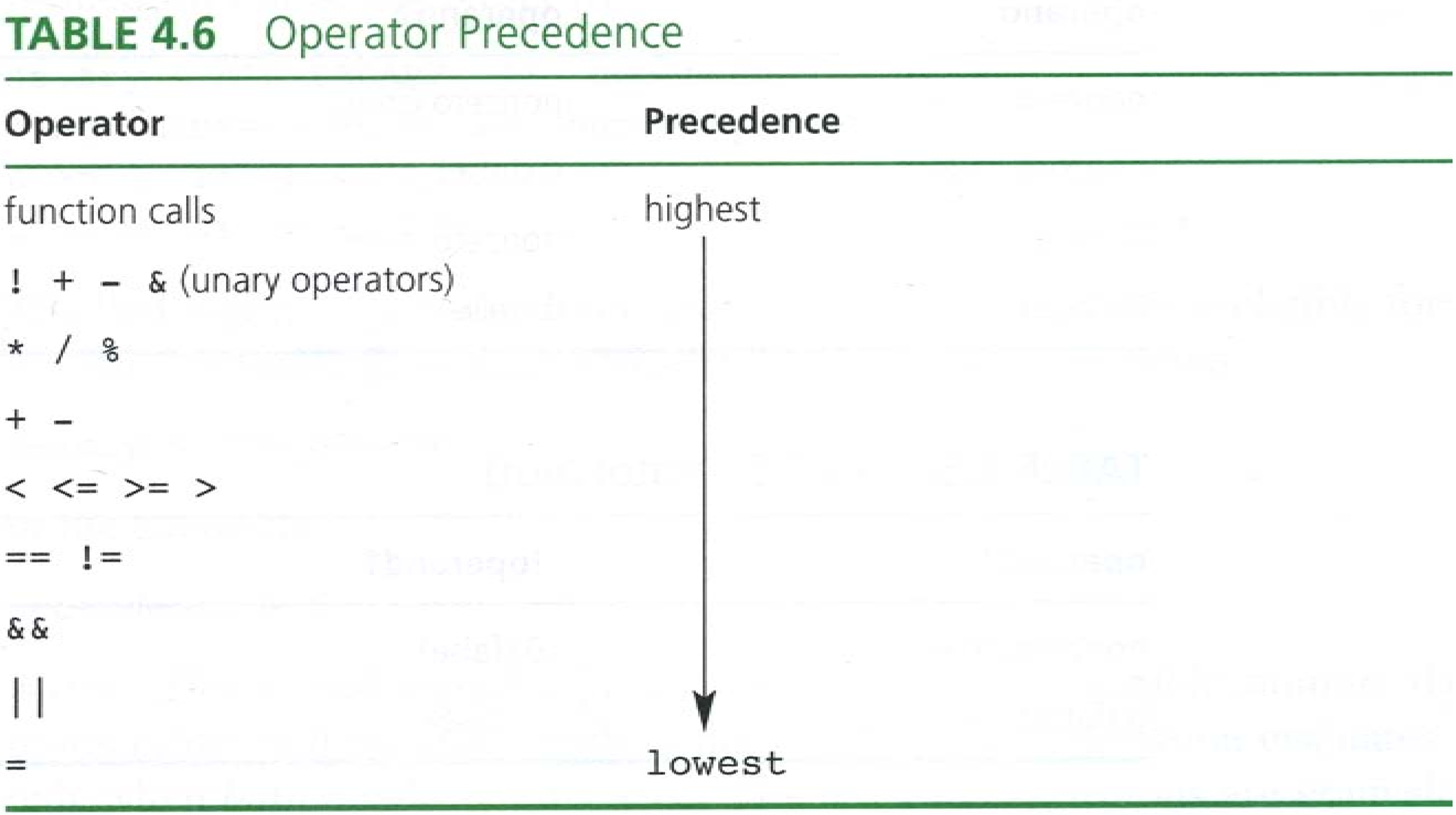
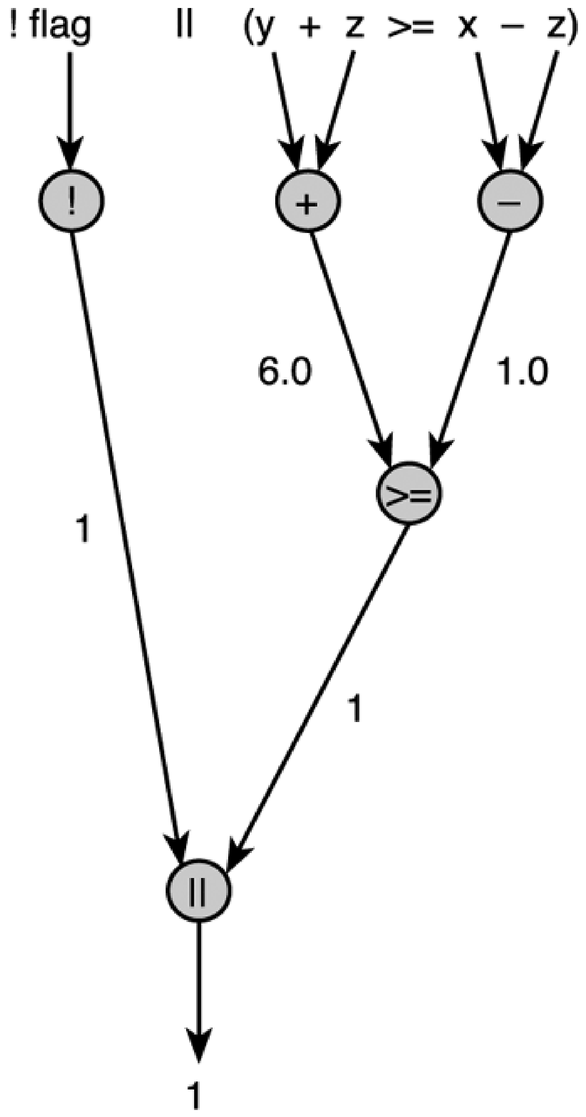
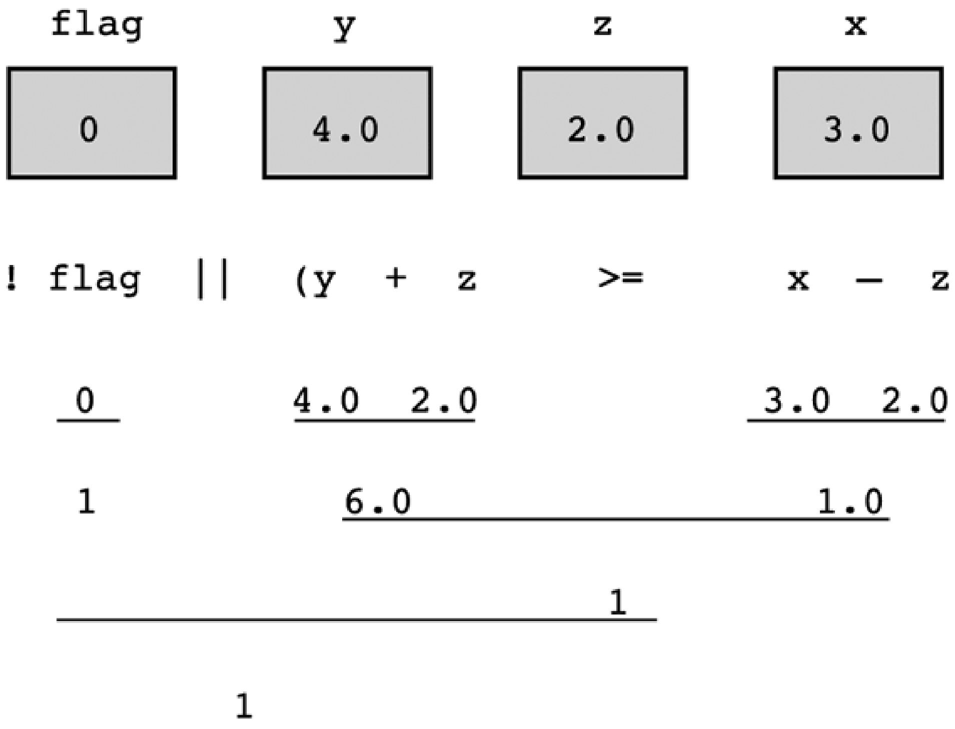


TABLE 4.6 Operator Precedence

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

Evaluation for !flag || (y + z >= x - z)



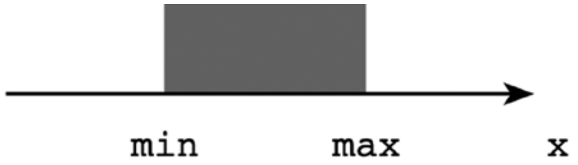


Short-Circuit Evaluation

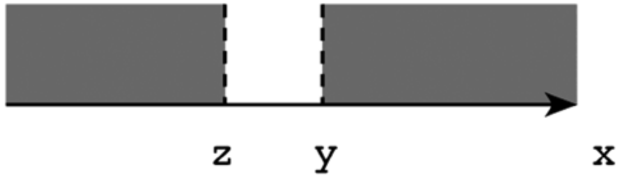
- For logical && and || operations C evaluates the left operand first and right operand later
- C stops evaluation
  - If the operation is && and left operand is false
    - Value of the expression is false
  - If the operation is || and left operand is true
    - Value of the expression is true

Logical Expressions

- min <= x && x <= max



- z > x || x > y



- You can compare characters

  ‘a’ <= ch && ch <= ‘z’

- You can use DeMorgan’s Theorem for simplification

  !(‘a’ <= ch && ch <= ‘z’)

  ‘a’ > ch || ch > ‘z’

Logical Assignment

- Integers are used to represent logical values
  - non-zero value is true
  - zero is false

```
senior_citizen = (age >= 65);
not_senior_citizen = !senior_citizen;
male_senior_citizen = senior_citizen && gender == ‘M’;

is_letter = (‘a’ <= ch  &&  ch <= ‘z’) ||
            (‘A’ <= ch  &&  ch <= ‘Z’);

even = (n % 2 == 0)
```

The if statement

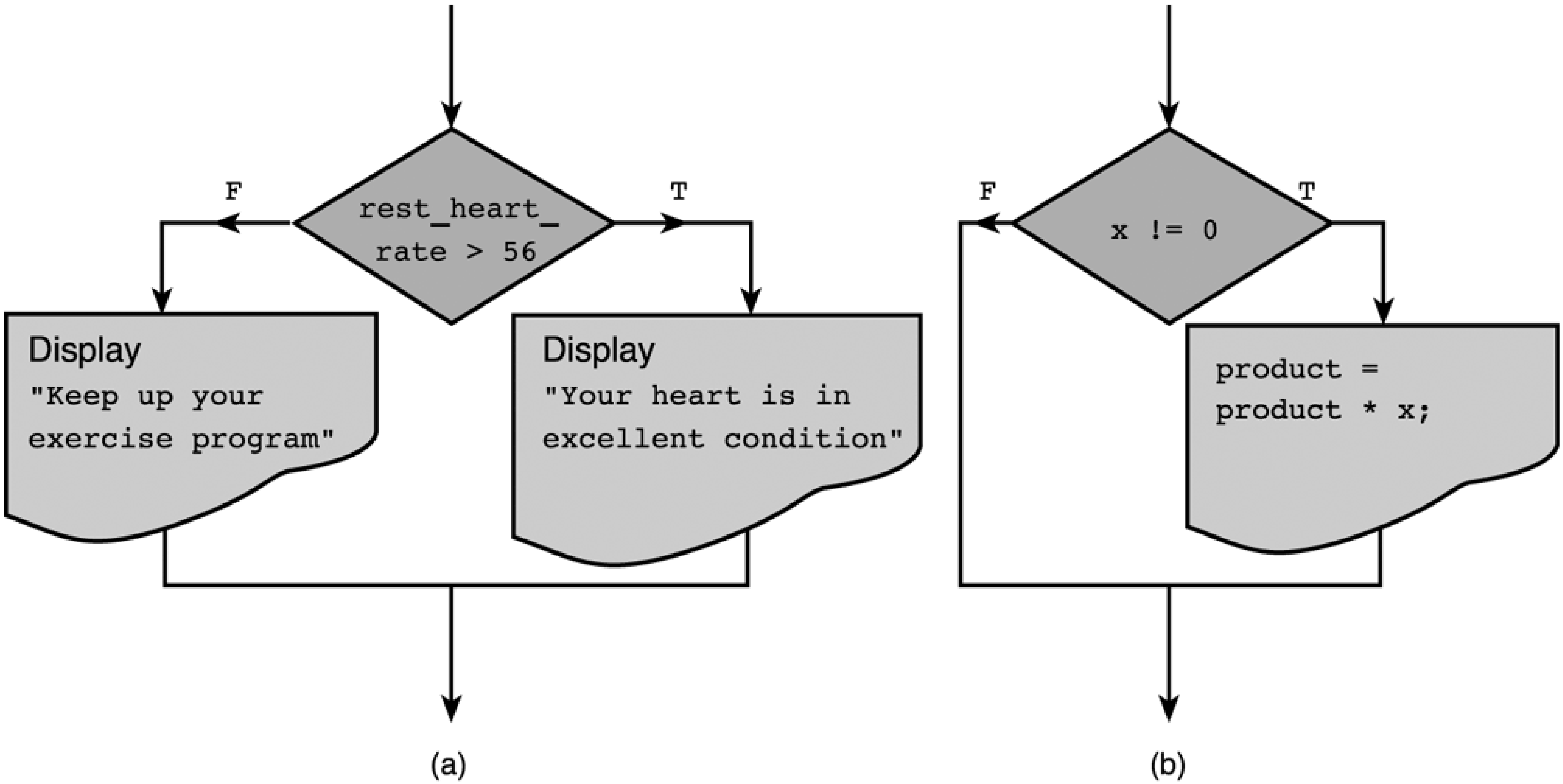
- if statement is the primary selection structure
- Two alternatives
  - Selects one of two alternative statement blocks

```
if (rest_heart_rate > 56)
   printf(“Keep up the exercise program! \n”);
else
   printf(“You heart is in excellent health! \n”);
```

- One alternative
  - Executes the statement block or not

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

Flowcharts of if Statements



(a) Two Alternatives and (b) One Alternative

The if statement

```
if (condition)
   statement;

if (condition)
   statement;
else
    statement;
```

```
if (x > 0)
    printf(“positive”);

 if (x > 0)
    printf(“positive”);
else
    printf(“negative”);
```

- What is the output?

```
if age > 65
   printf(“senior”);
printf(“citizen.\n”);
```

- What is the output?

```
if (age > 65);
   printf(“senior”);
printf(“citizen.\n”);
```

- What is the output?

```
if (age > 65) {
   printf(“senior”);
   printf(“citizen.\n”);
}
```

if statement with compound statements

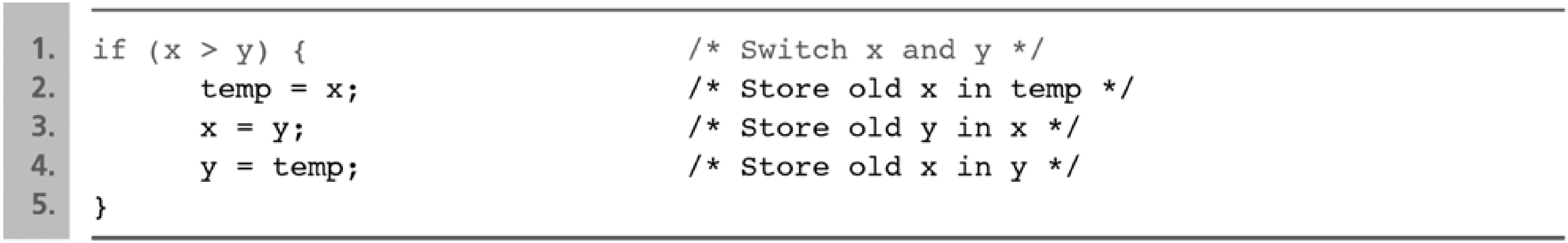
```
if (condition) {
    statements
}
```

```
if (condition) {
    statements
}
else {
    statements
}
```

```
if (radius > 0){
    circ = 2*PI*radius;
    printf(“%f”, circ);
}
```

```
if (radius > 0) {
    circ = 2*PI*radius;
    printf(“%f”, circ);
}
else {
    printf(“Radius is negative!..”);
}
```

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 */
}
```

Tracing an if statement

Hand trace = desk check

- To verify the correctness
- Step-by-step simulation of algorithm (or statements) on paper
  - Use simple input values
  - Trace each case
    - Try inputs that cause the condition to be false and true…
  - Execute each statement exactly as the computer
    - Don’t assume the way of execution
- Takes time
  - But saves time as well

Case Study: Simple Math Tool

Simple Math Tool to teach subtraction to a first grade student

- Algorithm
  1. Generate two single-digit integers randomly
     - number1 and number2 with number1 > number2
  2. Display the question
     - such as “What is 9 – 2?”
  3. Read student’s answer
  4. Display a message indicating whether the answer is correct

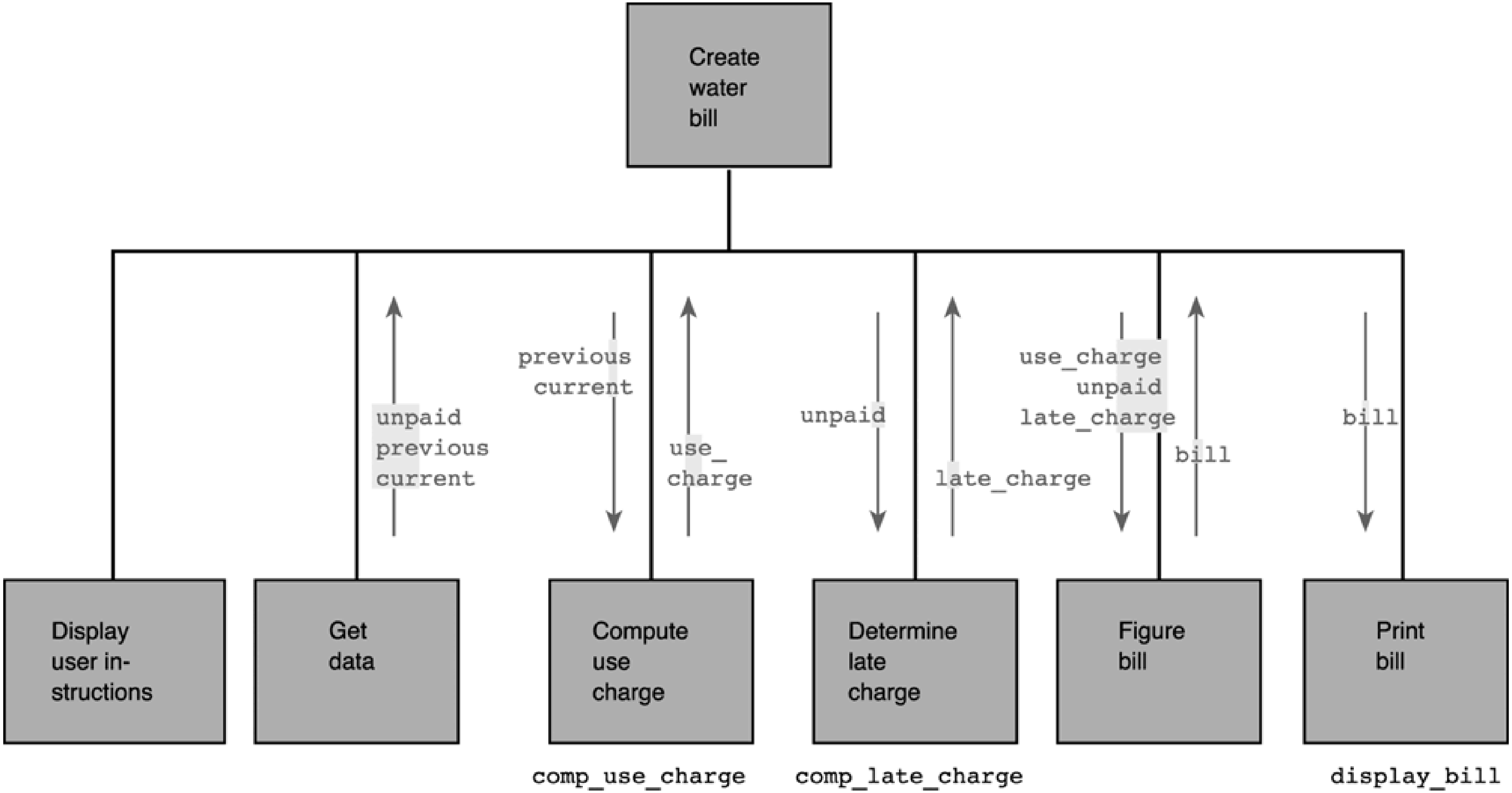
Case Study: Water Bill Problem

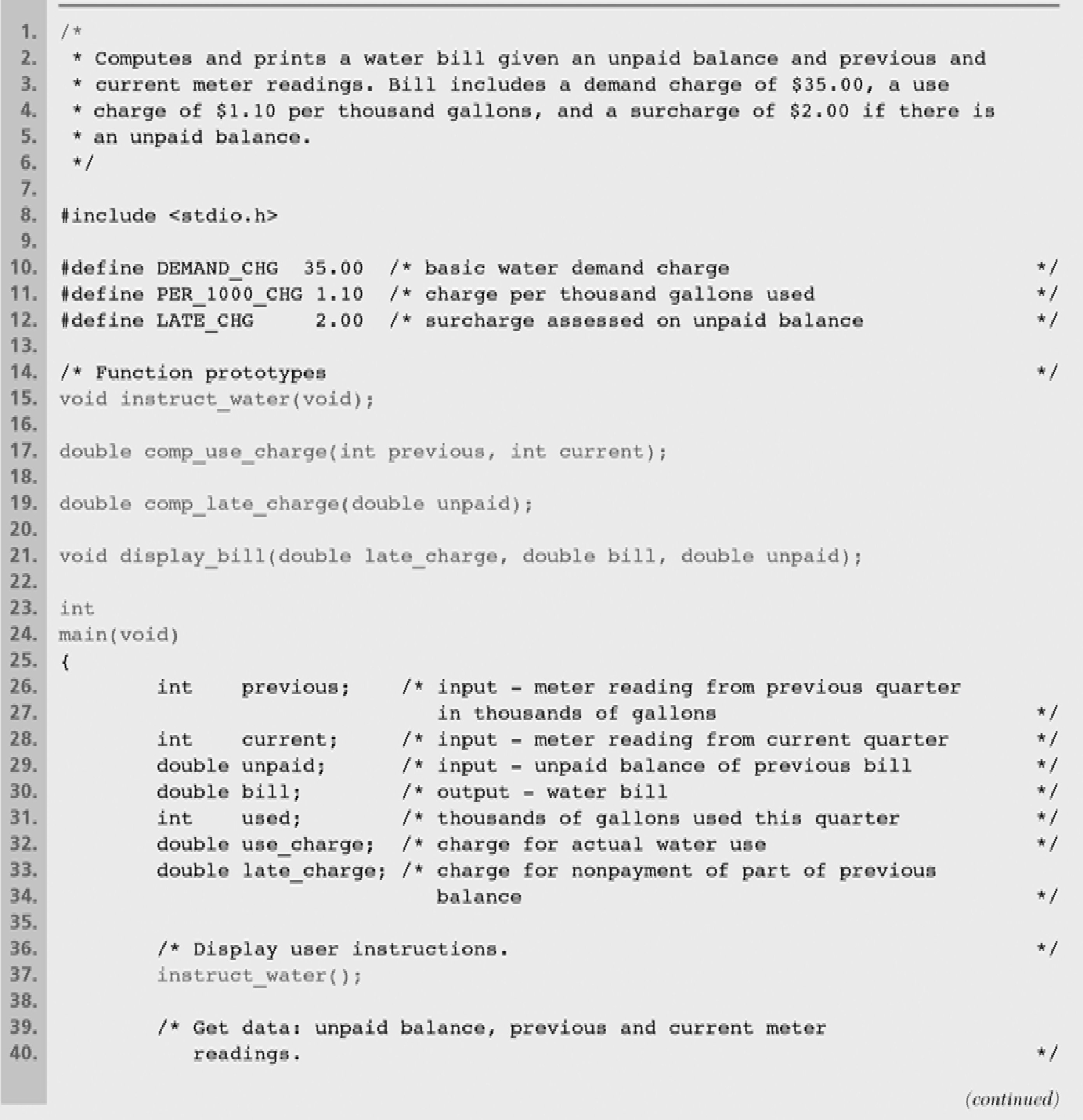
- Compute customers water bill
  - Demand charge = $35
  - Consumption charger = $1.10 per thousand gallons
  - Late charge for unpaid balance = $2
- Inputs:
  - Meter readings: previous, current
  - Unpaid balance
- Outputs:
  - Water bill : use charge, late chage

- Algorithm:
  1. Display user instructions
  2. Get data
  3. Compute use charge
  4. Determine late charge
  5. Figure bill amount
  6. Display the bill and charges

- Functions
  - Data requirements
  - Design and algorithm

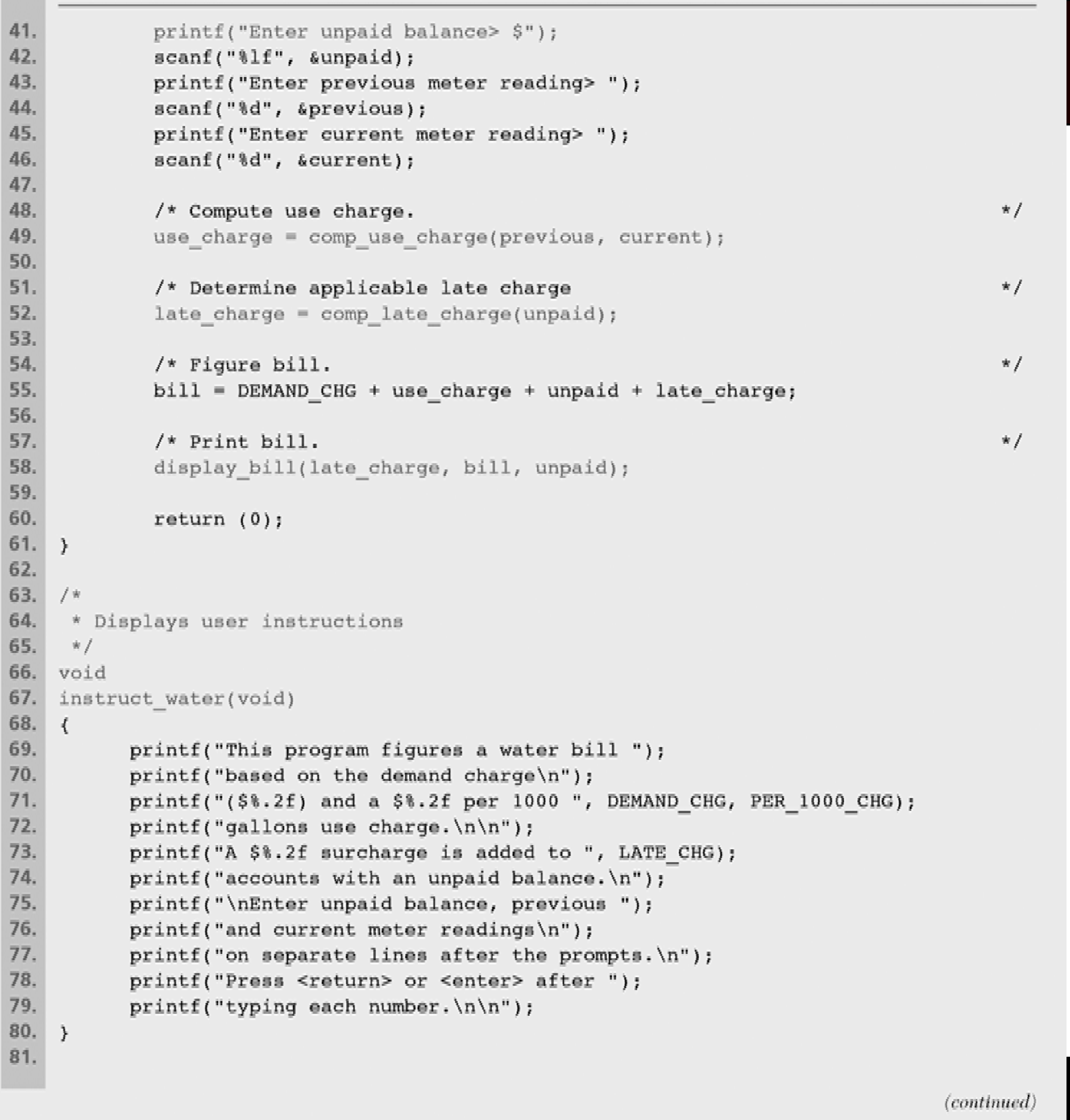
Structure Chart for Water Bill Problem





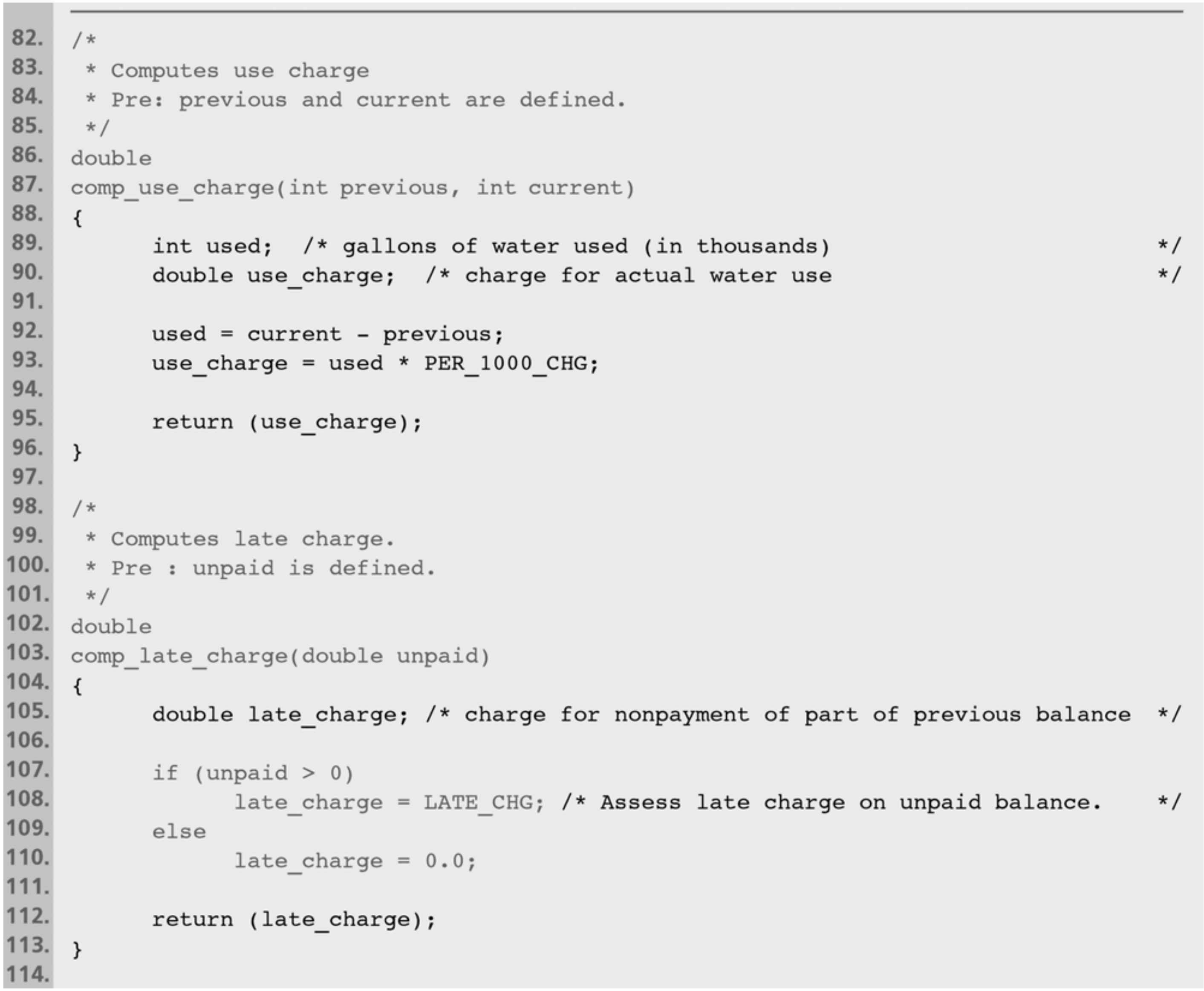
```c
/*
 * Computes and prints a water bill given an unpaid balance and previous and
 * current meter readings. Bill includes a demand charge of $35.00, a use
 * charge of $1.10 per thousand gallons, and a surcharge of $2.00 if there is
 * an unpaid balance.
 */

#include <stdio.h>

#define DEMAND_CHG  35.00  /* basic water demand charge                         */
#define PER_1000_CHG 1.10  /* charge per thousand gallons used                  */
#define LATE_CHG     2.00  /* surcharge assessed on unpaid balance              */

/* Function prototypes                                                          */
void instruct_water(void);

double comp_use_charge(int previous, int current);

double comp_late_charge(double unpaid);

void display_bill(double late_charge, double bill, double unpaid);

int
main(void)
{
        int    previous;    /* input - meter reading from previous quarter
                               in thousands of gallons                          */
        int    current;     /* input - meter reading from current quarter       */
        double unpaid;      /* input - unpaid balance of previous bill          */
        double bill;        /* output - water bill                              */
        int    used;        /* thousands of gallons used this quarter           */
        double use_charge;  /* charge for actual water use                      */
        double late_charge; /* charge for nonpayment of part of previous
                               balance                                          */

        /* Display user instructions.                                           */
        instruct_water();

        /* Get data: unpaid balance, previous and current meter
           readings.                                                            */
```

(continued)

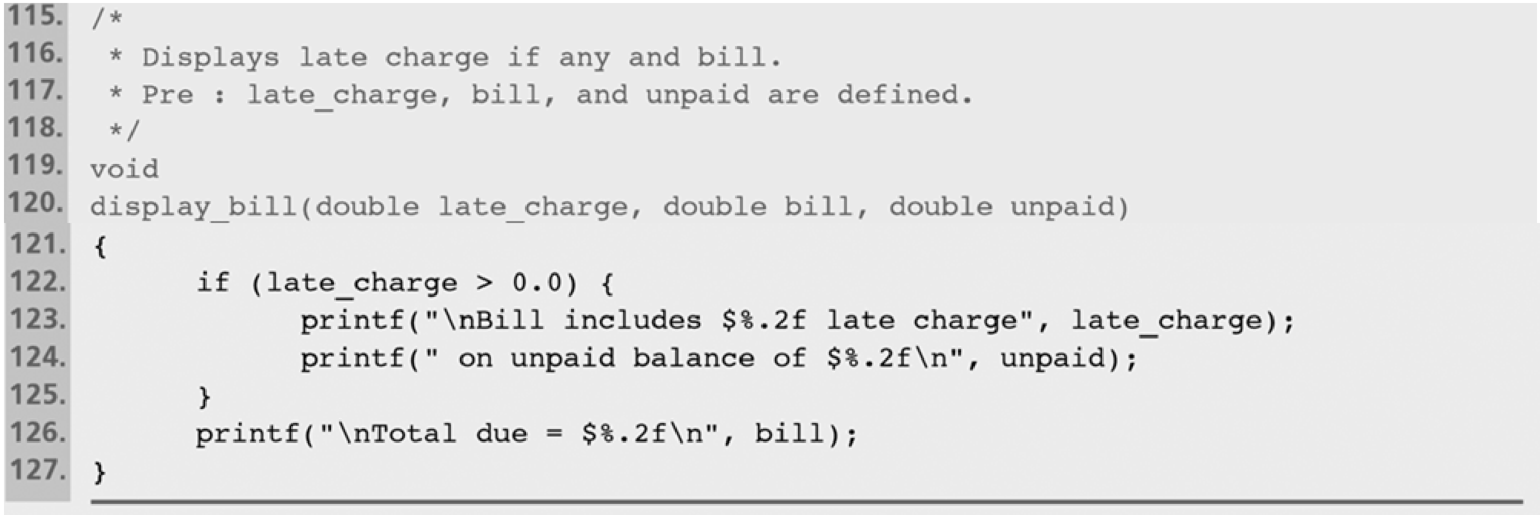


```c
        printf("Enter unpaid balance> $");
        scanf("%lf", &unpaid);
        printf("Enter previous meter reading> ");
        scanf("%d", &previous);
        printf("Enter current meter reading> ");
        scanf("%d", &current);

        /* Compute use charge.                                                 */
        use_charge = comp_use_charge(previous, current);

        /* Determine applicable late charge                                    */
        late_charge = comp_late_charge(unpaid);

        /* Figure bill.                                                        */
        bill = DEMAND_CHG + use_charge + unpaid + late_charge;

        /* Print bill.                                                         */
        display_bill(late_charge, bill, unpaid);

        return (0);
}

/*
 * Displays user instructions
 */
void
instruct_water(void)
{
      printf("This program figures a water bill ");
      printf("based on the demand charge\n");
      printf("($%.2f) and a $%.2f per 1000 ", DEMAND_CHG, PER_1000_CHG);
      printf("gallons use charge.\n\n");
      printf("A $%.2f surcharge is added to ", LATE_CHG);
      printf("accounts with an unpaid balance.\n");
      printf("\nEnter unpaid balance, previous ");
      printf("and current meter readings\n");
      printf("on separate lines after the prompts.\n");
      printf("Press <return> or <enter> after ");
      printf("typing each number.\n\n");
}

```

(continued)

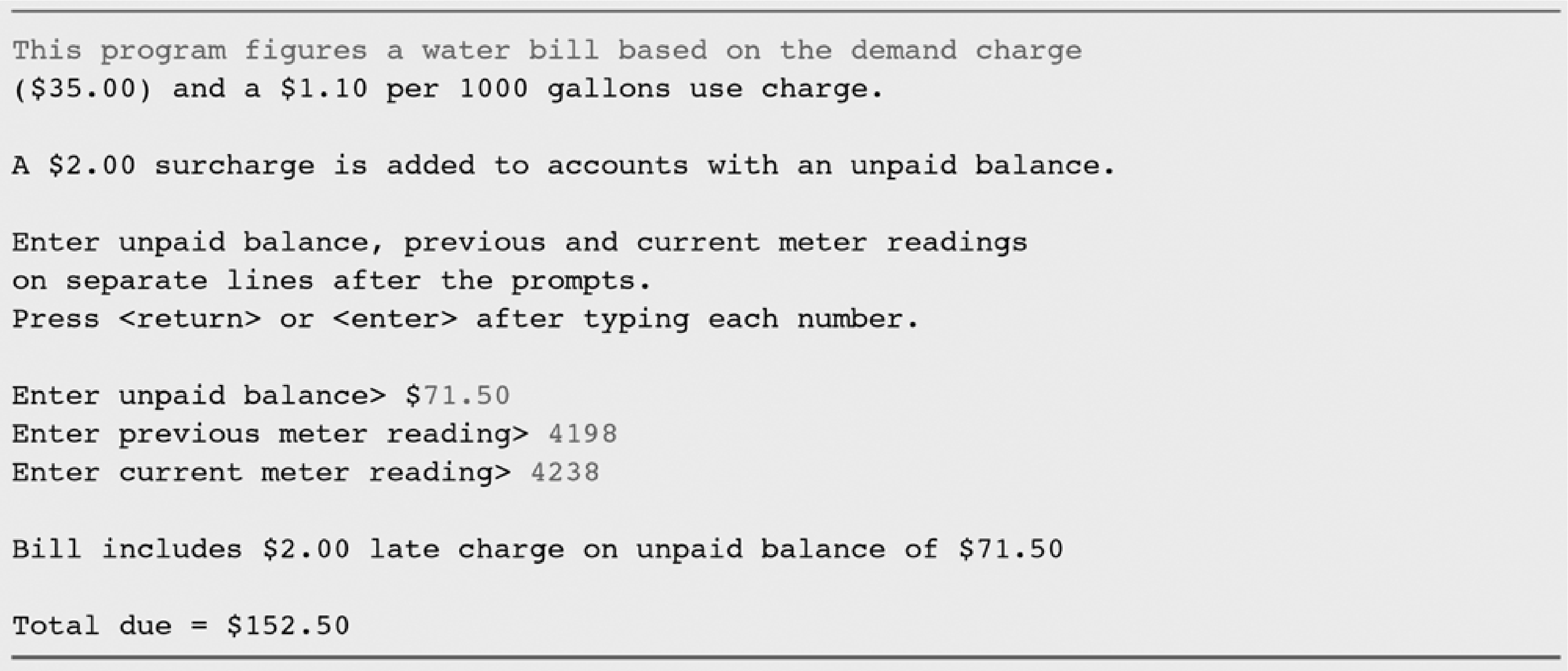


```c
/*
 * Computes use charge
 * Pre: previous and current are defined.
 */
double
comp_use_charge(int previous, int current)
{
      int used;   /* gallons of water used (in thousands)                       */
      double use_charge;  /* charge for actual water use                        */

      used = current - previous;
      use_charge = used * PER_1000_CHG;

      return (use_charge);
}

/*
 * Computes late charge.
 * Pre : unpaid is defined.
 */
double
comp_late_charge(double unpaid)
{
      double late_charge; /* charge for nonpayment of part of previous balance  */

      if (unpaid > 0)
            late_charge = LATE_CHG; /* Assess late charge on unpaid balance.    */
      else
            late_charge = 0.0;

      return (late_charge);
}

```



```c
/*
 * Displays late charge if any and bill.
 * Pre : late_charge, bill, and unpaid are defined.
 */
void
display_bill(double late_charge, double bill, double unpaid)
{
      if (late_charge > 0.0) {
            printf("\nBill includes $%.2f late charge", late_charge);
            printf(" on unpaid balance of $%.2f\n", unpaid);
      }
      printf("\nTotal due = $%.2f\n", bill);
}
```

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
```

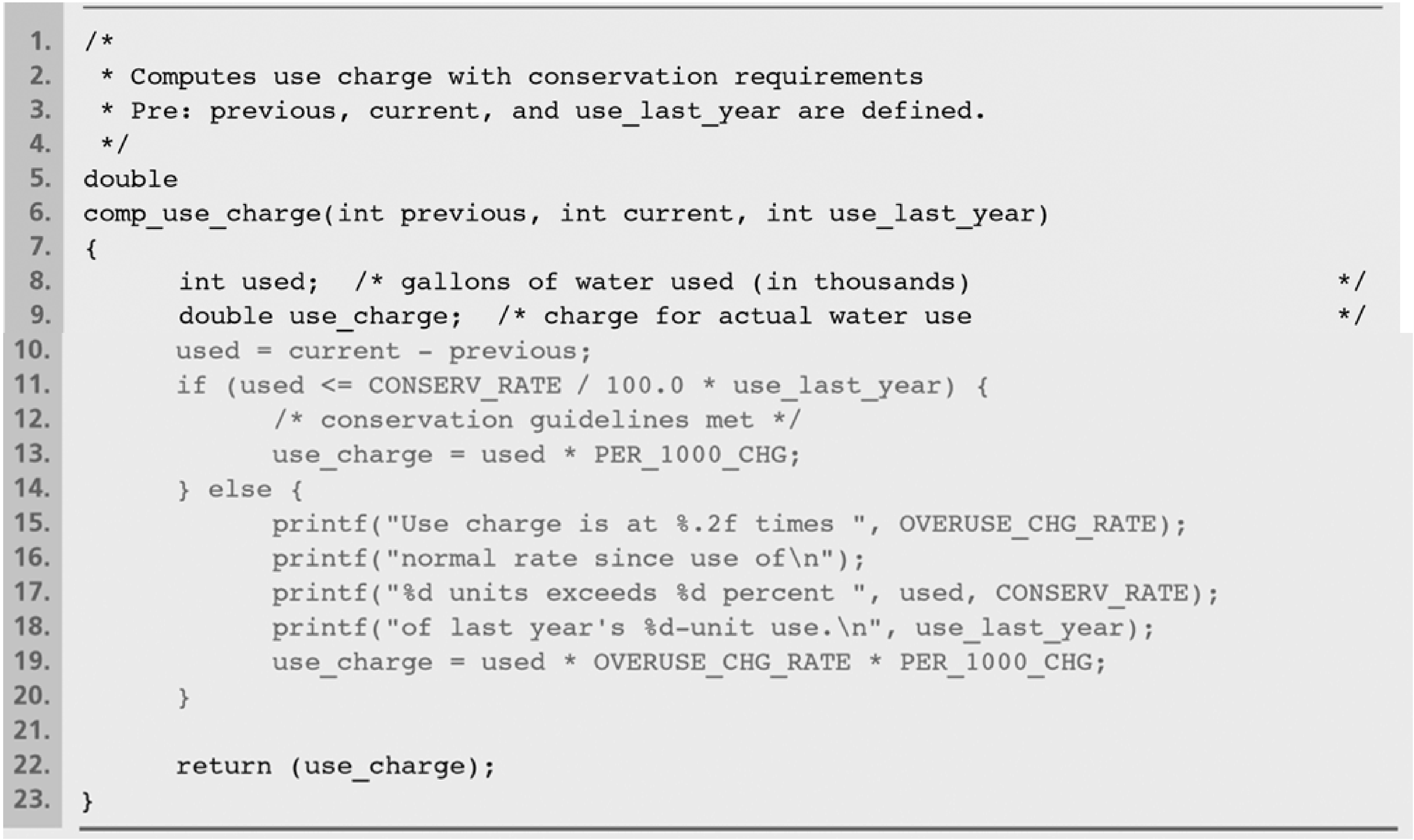
Program Style

- Consistent use of names in functions
  - Use same names to reference the same information
  - Ex: late_charge in three functions
    - They are all different variables but same information
- Cohesive functions
  - Each function should perform single operation
  - Easier to read, write, debug and maintain
  - More reusable
- Use constant macros
  - Can be used anywhere in the same file
  - Statements are easier to understand (more descriptive)
  - Easier to maintain

Case Study: Water bill with conservation requirement

- Modify the program
  - Conservation requirement: 5% decrease each year
  - Charge twice if more than %95 of the last year
- What changes are required?

Function comp_use_charge Revised



```c
/*
 * Computes use charge with conservation requirements
 * Pre: previous, current, and use_last_year are defined.
 */
double
comp_use_charge(int previous, int current, int use_last_year)
{
      int used;  /* gallons of water used (in thousands)                       */
      double use_charge;  /* charge for actual water use                       */
      used = current - previous;
      if (used <= CONSERV_RATE / 100.0 * use_last_year) {
            /* conservation guidelines met */
            use_charge = used * PER_1000_CHG;
      } else {
            printf("Use charge is at %.2f times ", OVERUSE_CHG_RATE);
            printf("normal rate since use of\n");
            printf("%d units exceeds %d percent ", used, CONSERV_RATE);
            printf("of last year's %d-unit use.\n", use_last_year);
            use_charge = used * OVERUSE_CHG_RATE * PER_1000_CHG;
      }

      return (use_charge);
}
```

Nested if statements

- if statement in another if statement
- Used if there are more than one alternative decisions

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

Alternative ways

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

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

- Less efficient
- Less readable

Alternative ways

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

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

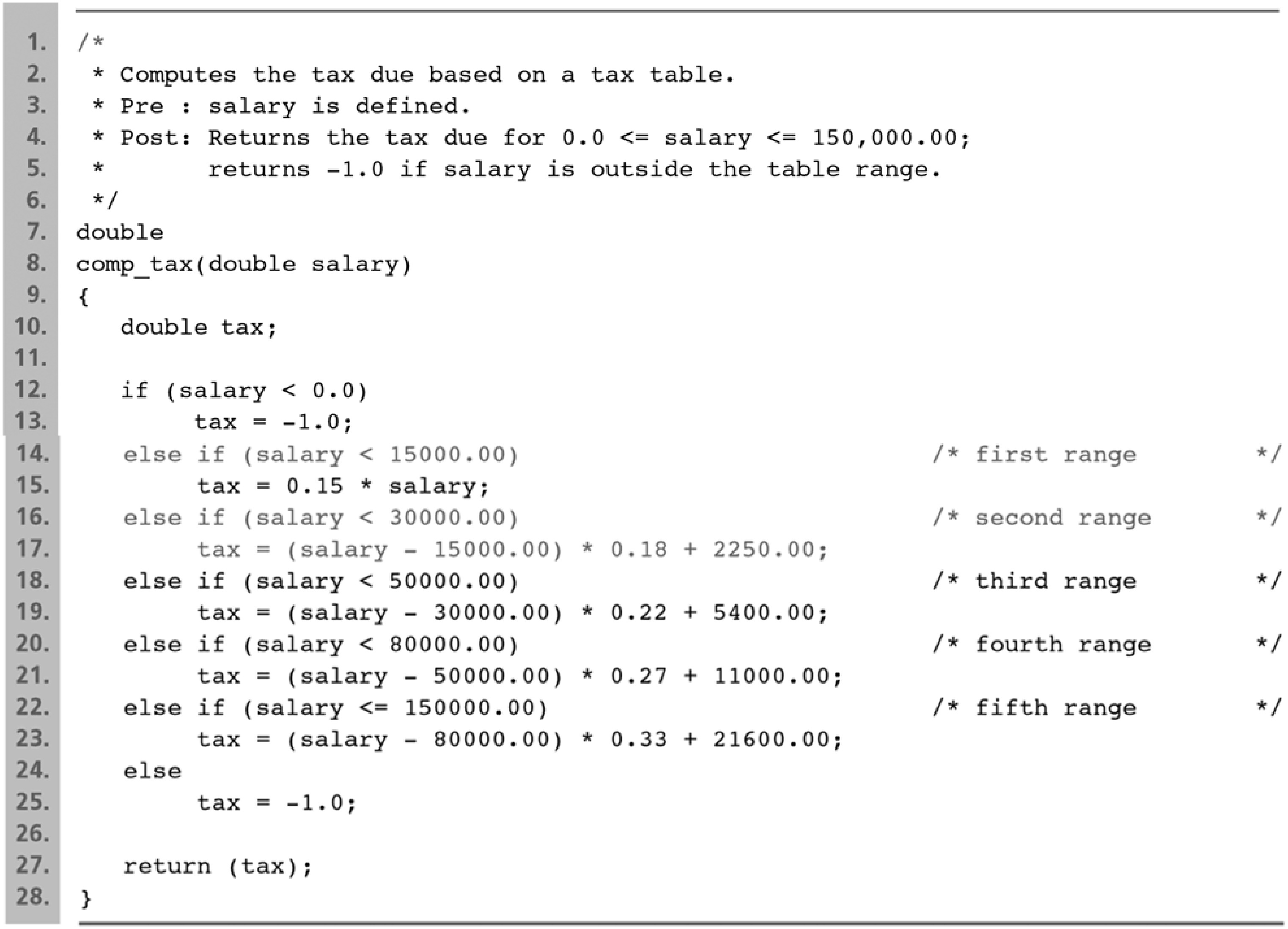
- Better way writing

Example: Payroll system

- Compute tax amount for a salary
- Decision table:

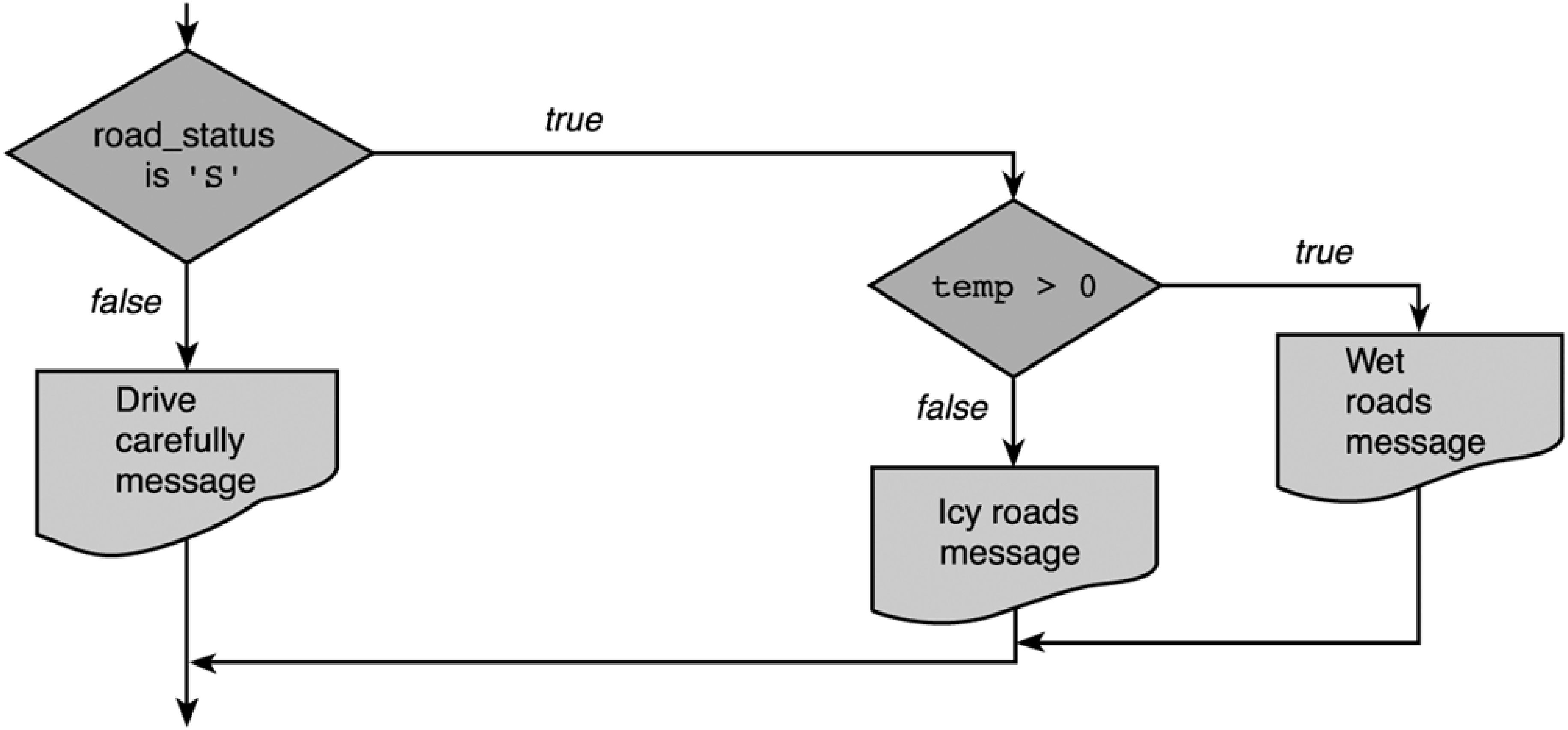
Salary	Tax rate
0 – 15,000	15
15,000 – 30,000	18
30,000 – 50,000	22
50,000 – 80,000	27
80,000 – 150,000	33

Function comp_tax



```c
/*
 * Computes the tax due based on a tax table.
 * Pre : salary is defined.
 * Post: Returns the tax due for 0.0 <= salary <= 150,000.00;
 *       returns -1.0 if salary is outside the table range.
 */
double
comp_tax(double salary)
{
   double tax;

   if (salary < 0.0)
        tax = -1.0;
   else if (salary < 15000.00)                            /* first range        */
        tax = 0.15 * salary;
   else if (salary < 30000.00)                            /* second range       */
        tax = (salary - 15000.00) * 0.18 + 2250.00;
   else if (salary < 50000.00)                            /* third range        */
        tax = (salary - 30000.00) * 0.22 + 5400.00;
   else if (salary < 80000.00)                            /* fourth range       */
        tax = (salary - 50000.00) * 0.27 + 11000.00;
   else if (salary <= 150000.00)                          /* fifth range        */
        tax = (salary - 80000.00) * 0.33 + 21600.00;
   else
        tax = -1.0;

   return (tax);
}
```

Flowchart of Road Sign Decision



```
if (road_status == ‘S’)
    if (temp > 0) {
           printf(“wet road”);
    } else {
           printf(“icy road”);
    }
else
    printf(“drive carefully”);
```

```
if (road_status == ‘S’)
    if (temp > 0) {
           printf(“wet road”);
    }
else
    printf(“drive carefully”);
```

```
if (road_status == ‘S’)
    if (temp > 0) {
            printf(“wet road”);
    } else {
            printf(“icy road”);
    }
else
    printf(“drive carefully”);
```

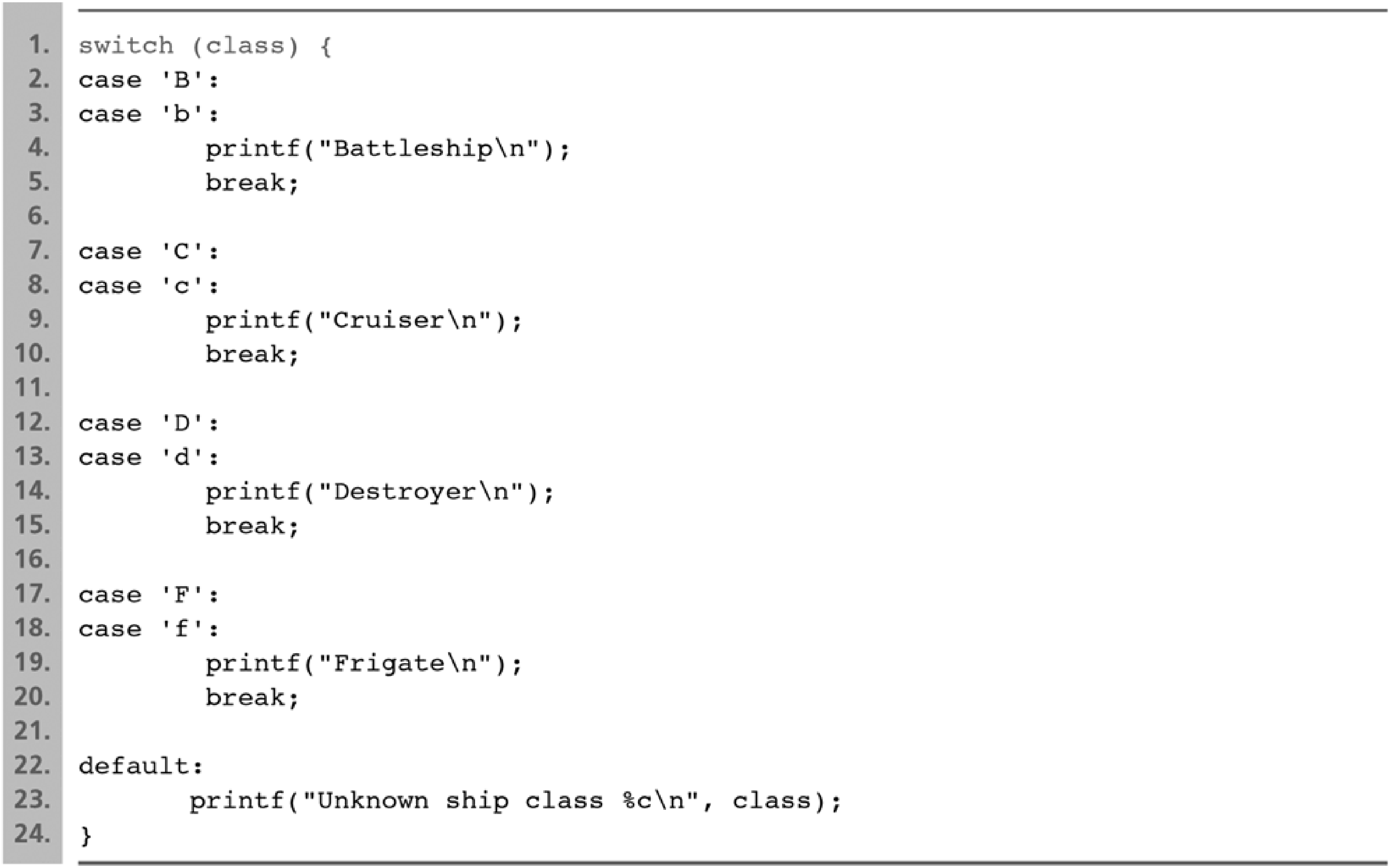
```
if (road_status == ‘S’){
    if (temp > 0) {
            printf(“wet road”);
    }
} else
    printf(“drive carefully”);
```

- C associates an else with the most recent if statement
- Use braces to force association

The switch statement

- Select one of the several alternatives
  - Selection is based on the value of a single variable (of type int of char not double)

Example of a switch Statement



```
switch (class) {
case 'B':
case 'b':
        printf("Battleship\n");
        break;

case 'C':
case 'c':
        printf("Cruiser\n");
        break;

case 'D':
case 'd':
        printf("Destroyer\n");
        break;

case 'F':
case 'f':
        printf("Frigate\n");
        break;

default:
       printf("Unknown ship class %c\n", class);
}
```

The switch statement

```
switch (controlling expression) {
    label set1:
          statements;
          break;
    label set2:
          statements;
          break;
    . . . .
    label setn:
          statements;
          break;
    default:
          statements;
    }
```

The switch statement

- Statements following the matching case label are executed until a break statement
  - After the break the rest of the switch statement is skipped
- If no case label matches statements after the default label are executed

- The switch statement is more readable
- Try to use default case

switch without break

```
switch(Grade) {
   case 'A' : printf("Excellent\n");
   case 'B' : printf("Good\n" );
   case 'C' : printf("OK\n" );
   case 'D' : printf("Mmmmm....\n");
   case 'F' : printf("You must do better than this\n");
   default  : printf("What is your grade anyway?\n");
}
```

For instance when Grade is ‘A’, the output is:

```
Excellent
Good
OK
Mmmmm....
You must do better than this
What is your grade anyway?
```

switch with break

```
switch(Grade) {
   case 'A' : printf("Excellent\n");
              break;

   case 'B' : printf("Good\n");
              break;

   case 'C' : printf("OK\n");
              break;

   case 'D' : printf("Mmmmm....\n");
              break;

   case 'F' : printf("You must do better than this\n");
              break;

   default :  printf("What is your grade anyway?\n");
}
```

For instance when Grade is ‘B’, the output is:

```
Good
```

Another switch example

```
switch (month) {
   case 1:
   case 3:
   case 5:
   case 7:
   case 8:
   case 10:
   case 12: numDays = 31;
             break;
   case 4:
   case 6:
   case 9:
   case 11: numDays = 30;
             break;
   case 2:  if((year % 4) == 0)
                 numDays = 29;
             else
                 numDays = 28;
             break;
   default: printf("You have entered a wrong month number.\n");
}
```