"Stupidity is while (1) { tryAgain(); }"

- Unknown

CSE102
Computer Programming with C

2016-2017 Fall Semester

Selection Structures: "if" and "switch"

Largely adapted from J.R. Hanly, E.B. Koffman, F.E. Sevilgen, and others…

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:

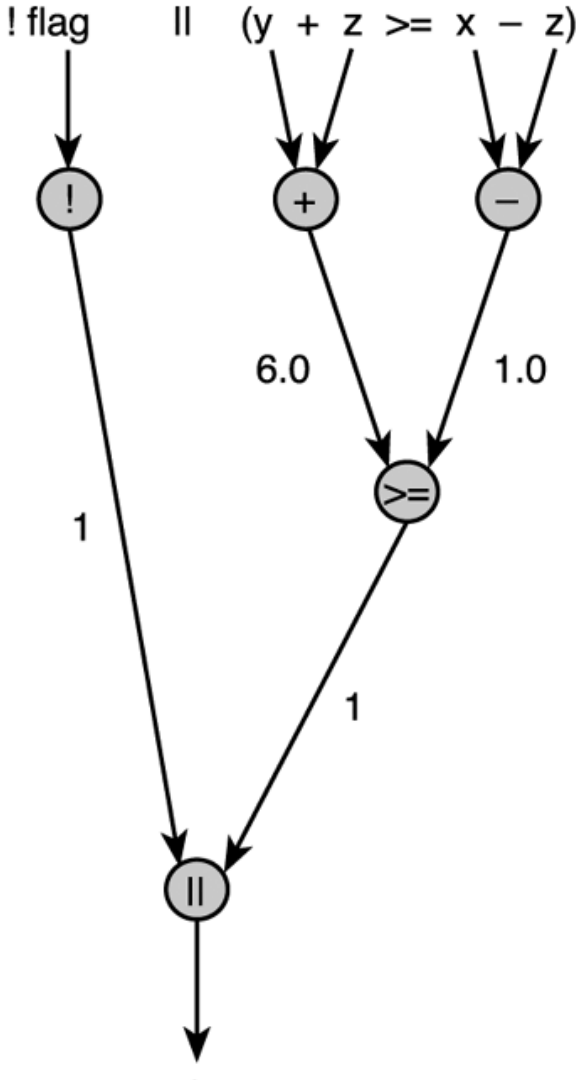
```
salary < MIN_SALARY || dependents > 5
temperature > 90.0 && humidity > 0.90
n >= 0 && n <= 100
!(n >= 0 && n <= 100)
```

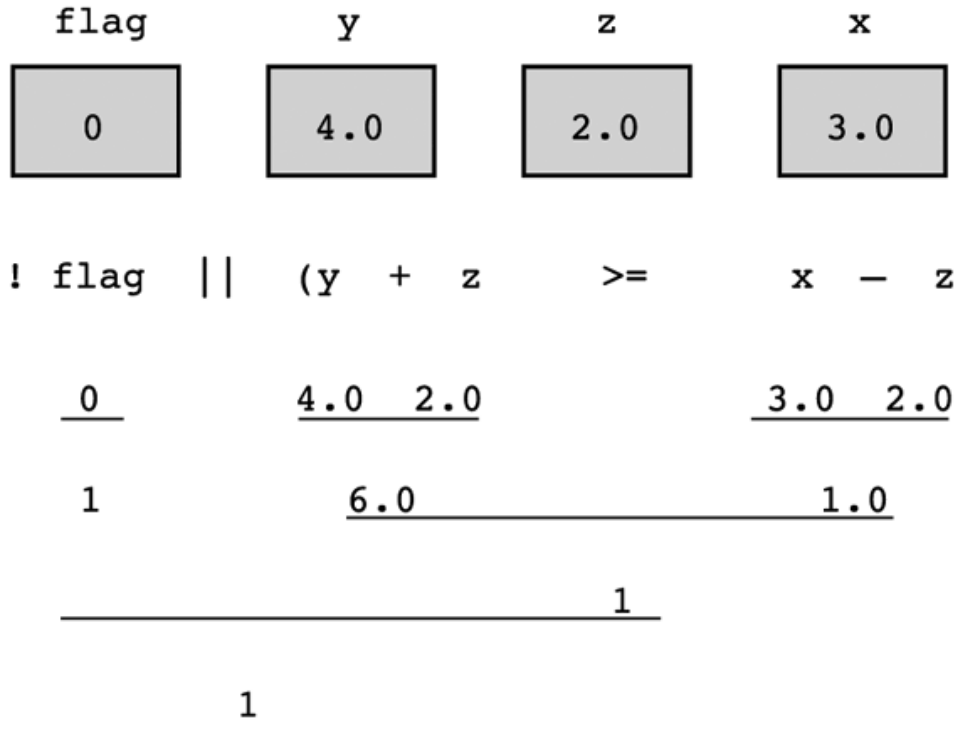
Operator Precedence

TABLE 4.6 Operator Precedence

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

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



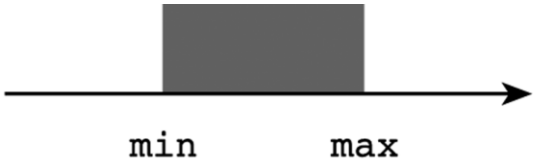


Short-Circuit Evaluation

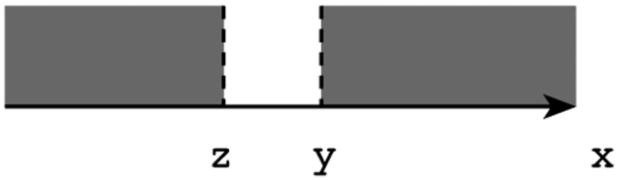
- For logical && and || operations C evaluates the left operand first and right operand later
- C stops evaluation
  - If the operation is && and left operand is false
    - Value of the expression is false
  - If the operation is || and left operand is true
    - Value of the expression is true

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

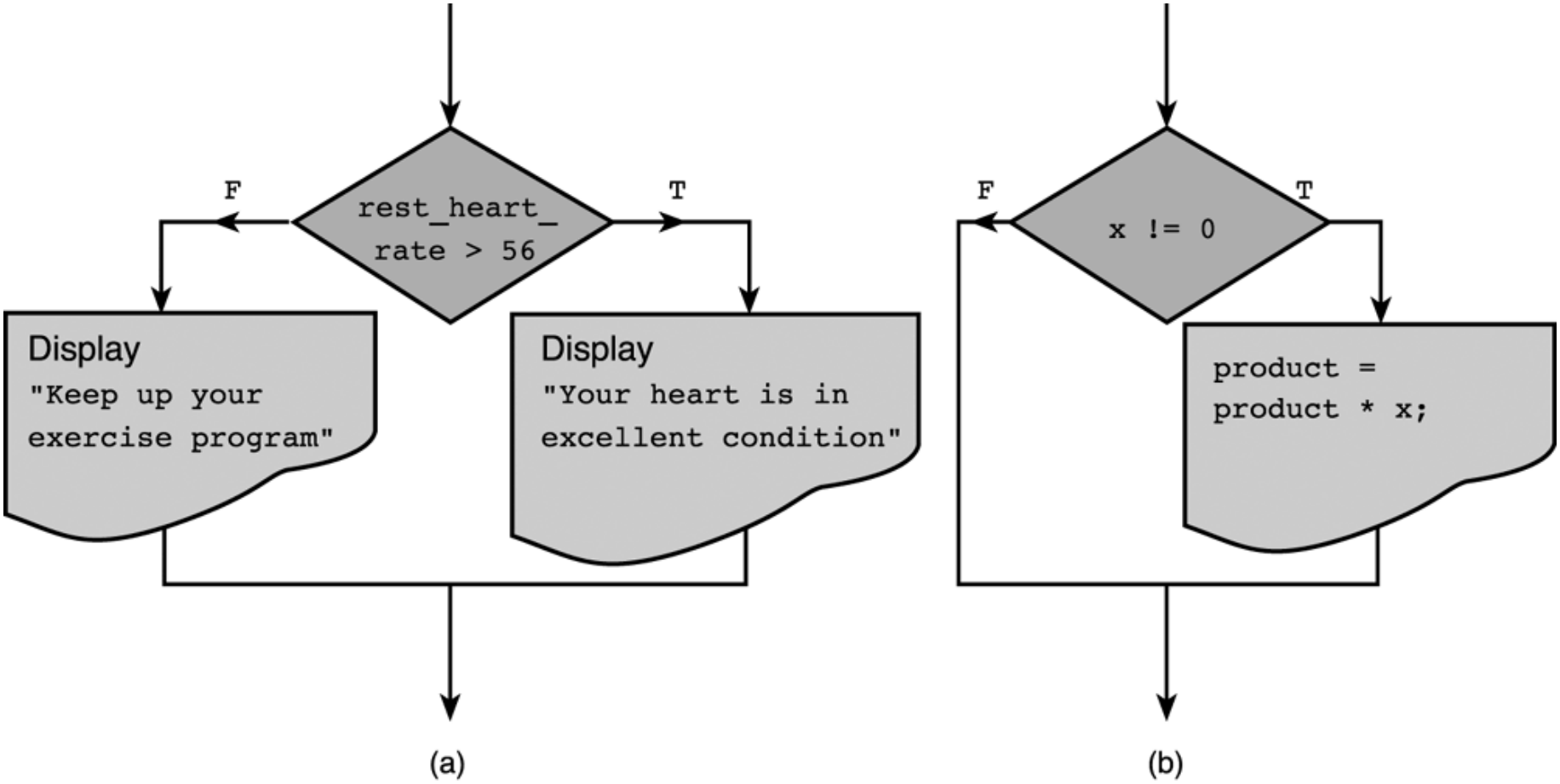
- if statement is the primary selection structure
- Two alternatives
  - Selects one of two alternative statement blocks

```
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

```
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. }
```

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

Hand Tracing an IF Statement

```
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.  }
```

Line No	x	y	temp
	5	3	
1			
2			5
3	3		
4		5	
5			

Hand Tracing an IF Statement

```
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.  }
```

Line No	x	y	temp
	1	3	
1			
2			
3			
4			
5			

Hand Tracing an IF Statement

```
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.  }
```

Line No	x	y	temp
	3	1	
1			
2			3
3	1		
4		3	
5			

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

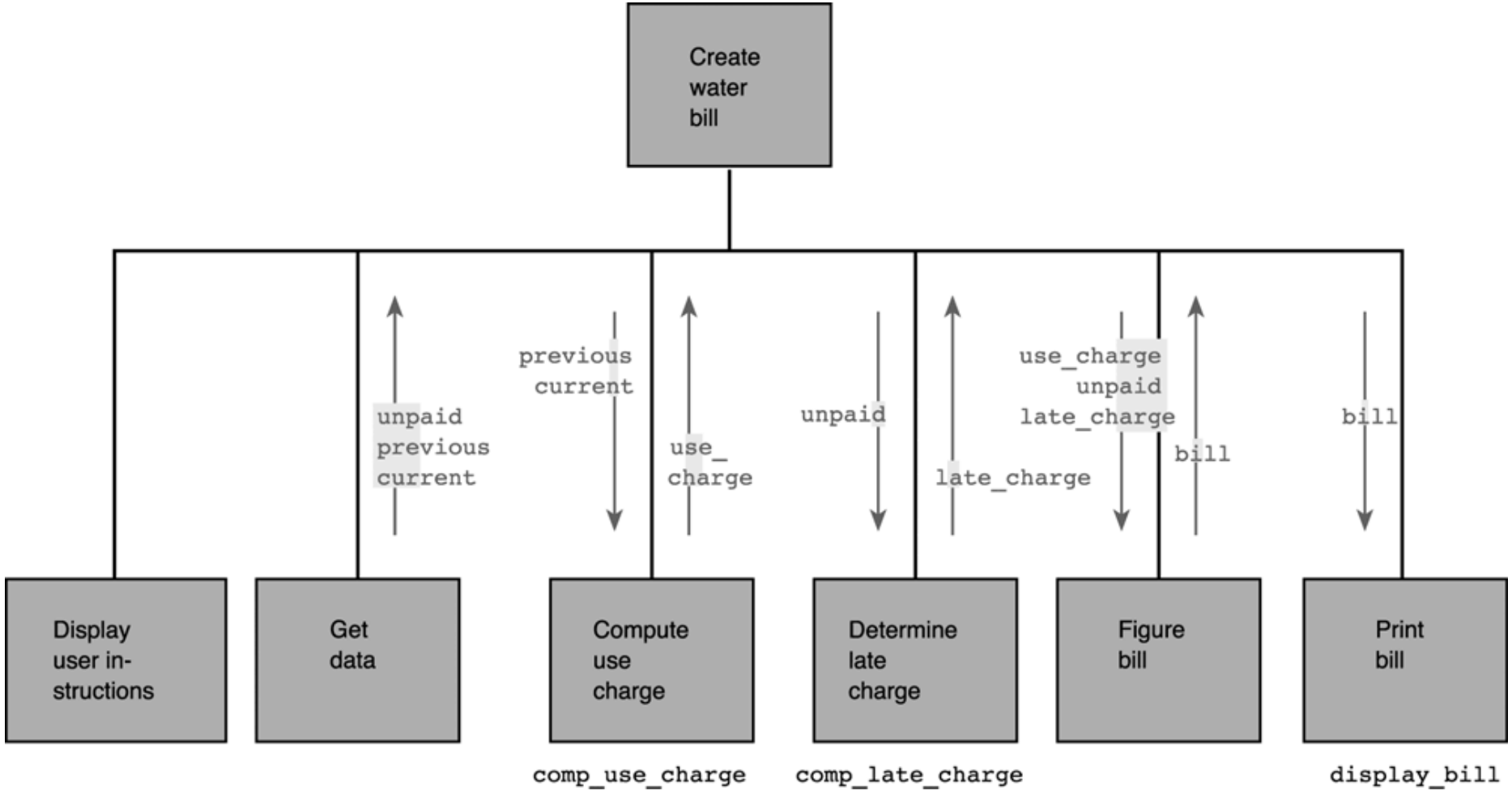
- Compute customers water bill
  - Demand charge = $35
  - Consumption charger = $1.10 per thousand gallons
  - Late charge for unpaid balance = $2
- Inputs:
  - Meter readings: previous, current
  - Unpaid balance
- Outputs:
  - Water bill : use charge, late chage

Water Bill Problem

- 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



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.
```

(continued)

Water Bill Problem

```
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)

Water Bill Problem

```
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.
```

(continued)

Water Bill Problem

```
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)

Water Bill Problem

```
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.
```

Water Bill Problem

```
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. }
```

Water Bill Problem

```
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)

```
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. }
```

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

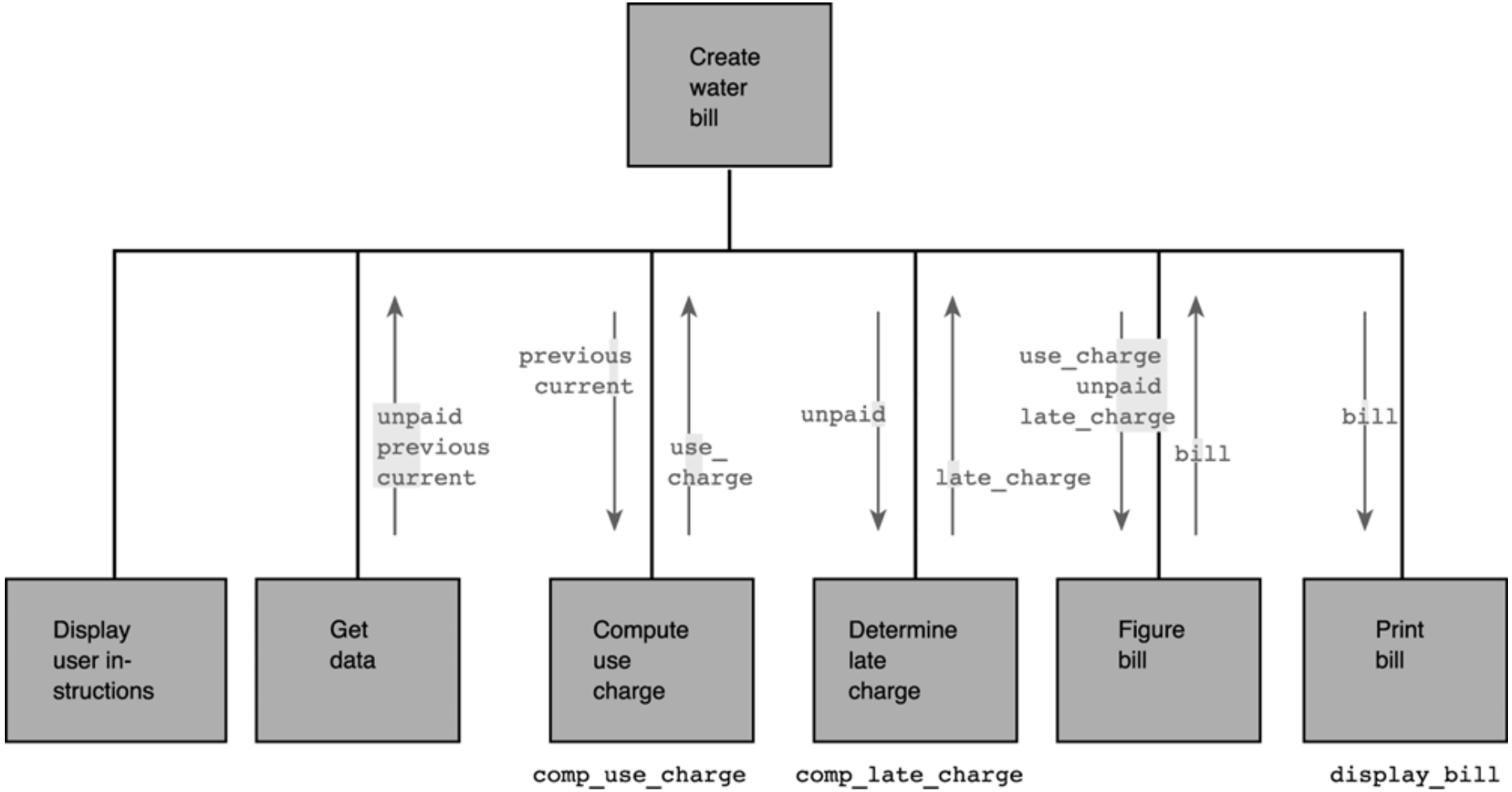
- Consistent use of names in functions
  - Use same names to reference the same information
  - Ex: late_charge in three functions
    - They are all different variables but same information
- Cohesive functions
  - Each function should perform single operation
  - Easier to read, write, debug and maintain
  - 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?

Structure Chart for Water Bill Problem



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                   */
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. }
```

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

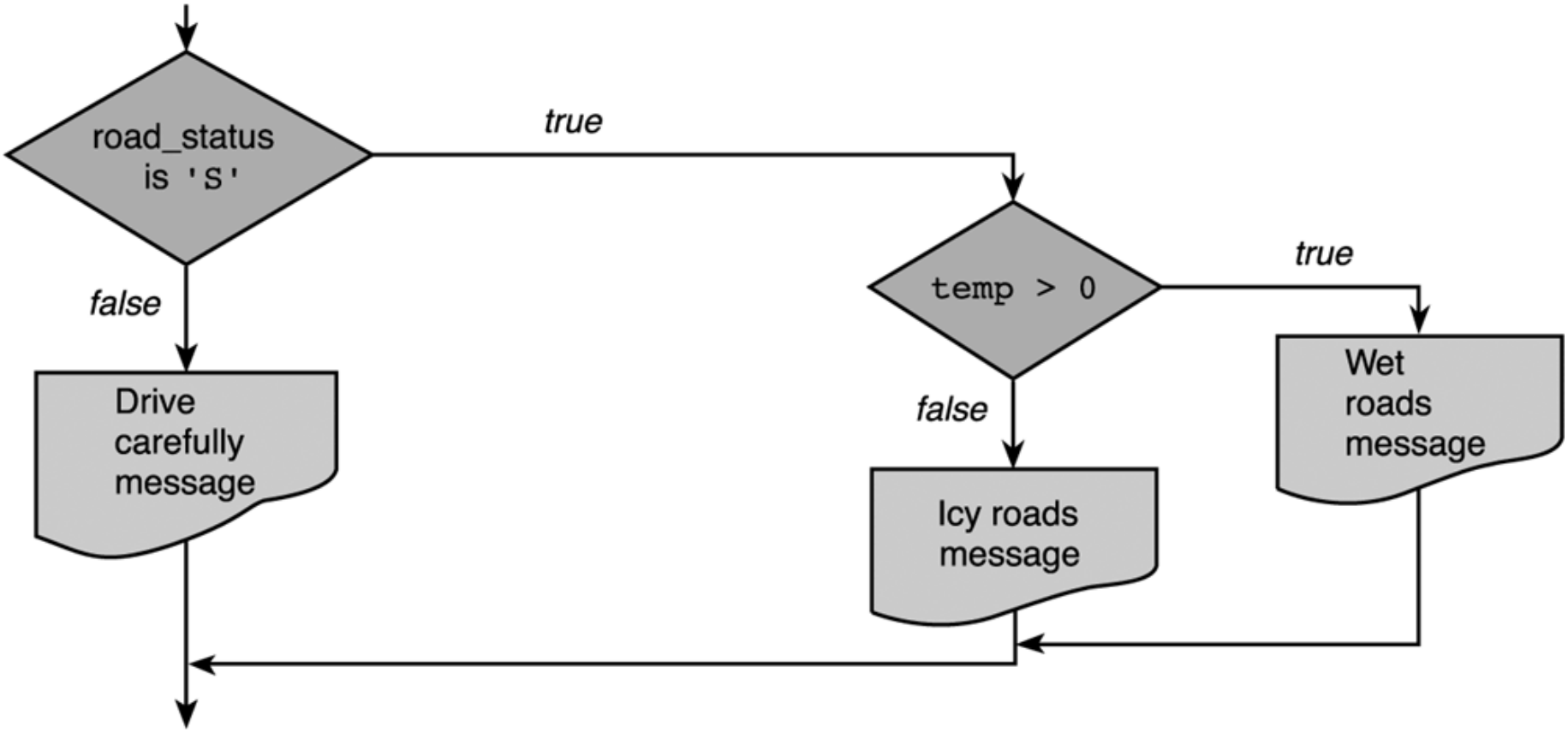
- Compute tax amount for a salary
- Decision table:

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

```
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;
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. }
```

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)

switch with break

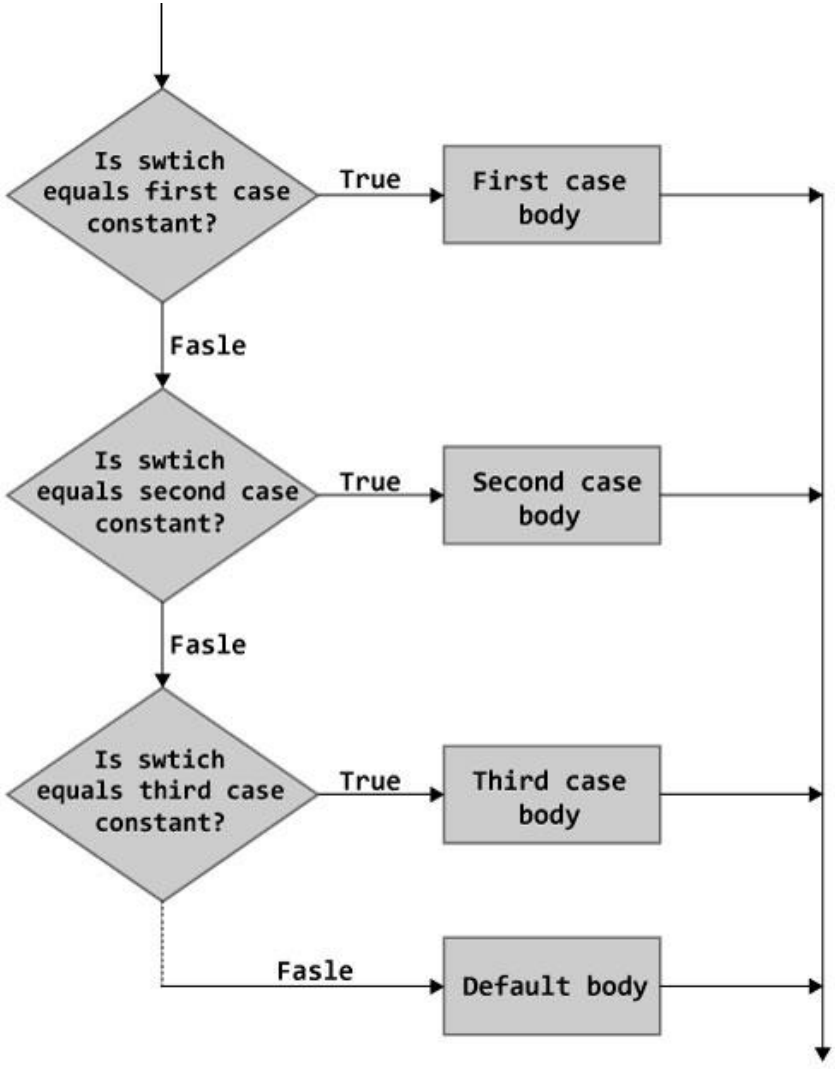
```
switch(Grade) {
   case 'A' : printf("Excellent\n");
              break;

   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
```

The switch statement

```
switch (controlling expression) {
    label case_1:
        statements;
        break;
    label case_2:
        statements;
        break;
    . . . .
    label case_n:
        statements;
        break;
    default:
        statements;
    }
```



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?
```

Example of a switch Statement

```
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

- 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

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");
}
```

Another switch example

```
/* Print the day of the week given a number between 1
 * and 7 where 1 is Monday */
void
print_day_of_week(int day)
{
   switch (day) {
       case 1: printf(“Monday”); break;
       case 2: printf(“Tuesday”); break;
       case 3: printf(“Wednesday”); break;
       case 4: printf(“Thursday”); break;
       case 5: printf(“Friday”); break;
       case 6: printf(“Saturday”); break;
       default: printf(“Sunday”);
}
```

Payroll System using Switch?

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