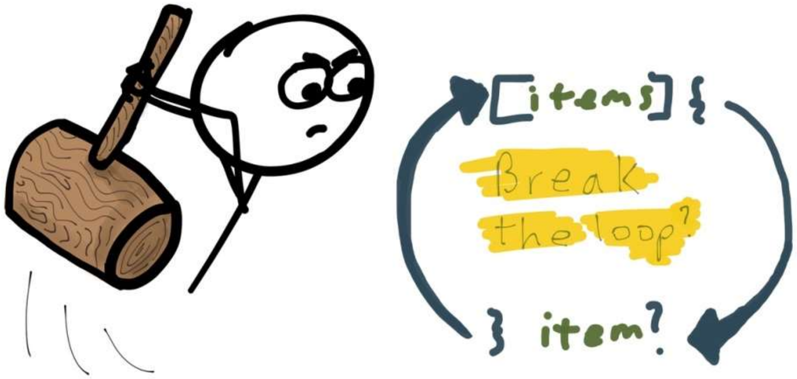


Repetition and Loop Statements

Mirza Mohammad Lutfe Elahi

Outline

- Repetition in Programs
- Counting loops
- The **`while`** statement
- The **`for`** statement
- Conditional Loops
- Nested Loops
- The **`do-while`** statement
- How to debug and test programs
- Common Programming Errors

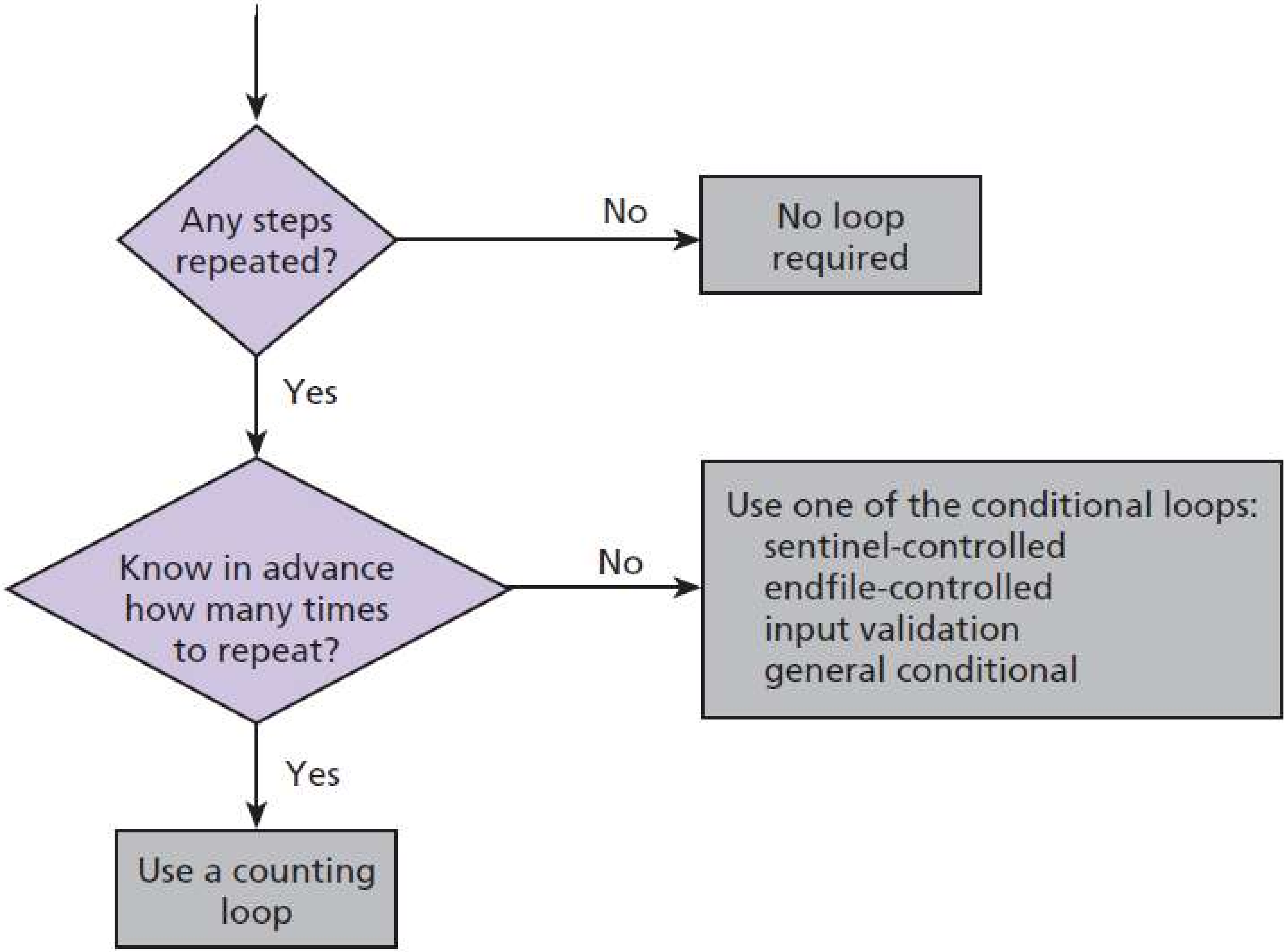
Recall: Control Structures

- Three kinds of control structures
  - **Sequence** (Compound Statement)
  - **Selection** (**`if`** and **`switch`** Statements)
  - **Repetition** (discussed in this presentation)
- The repetition of steps in a program is called a **loop**
- Three loop control structures in C
  - The **`while`** statement
  - The **`for`** statement
  - The **`do-while`** statement

Repetition in Programs

- **Loop structure**
  - A control structure that repeats a group of steps in a program
- **Loop body**
  - The statements that are repeated inside the loop
- Three questions to raise:
  1. Are there any steps repeated in the problem?
  2. If the answer to question 1 is yes, is the number of repetitions know in advance?
  3. If the answer to question 2 is no, then how long to keep repeating the steps?

Flowchart of Loop Choice



Counting Loop

- Called a **Counter-controlled loop**
- A loop that can be controlled by a **counter variable**
- Number of iterations (repetitions) can be determined before loop execution begins
- General format of a counting loop:

```
Set loop control variable to an initial value
while (loop control variable < final value) {
        /* Do something multiple times */
        Increase loop control variable by 1
}
```

The `while` Statement

- Syntax:

Loop Repetition Condition

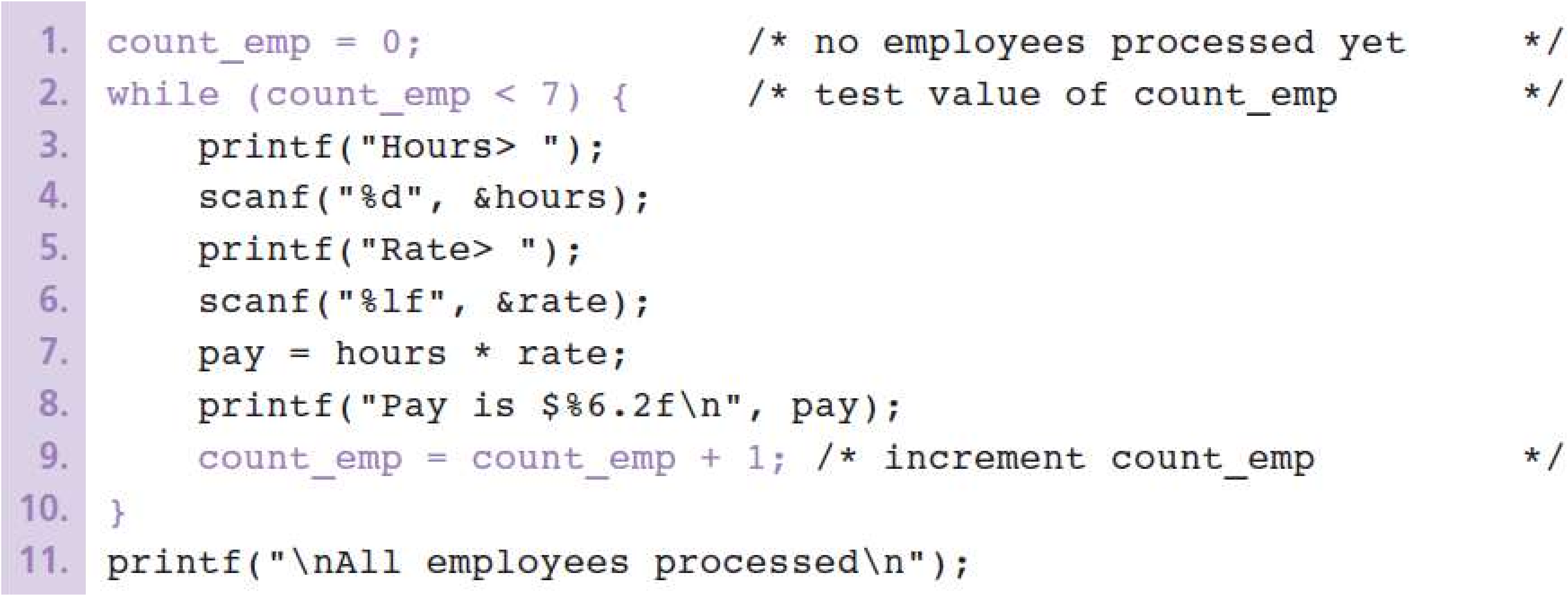
```
while (condition) {
   statement_1 ;
   statement_2 ;
   . . .
   statement_N ;
}
```

Loop Body:
Can be one statement, or
Compound statement

- As long as condition is true, the loop body is executed
- The condition is re-tested after each iteration
- The loop terminates when condition becomes false

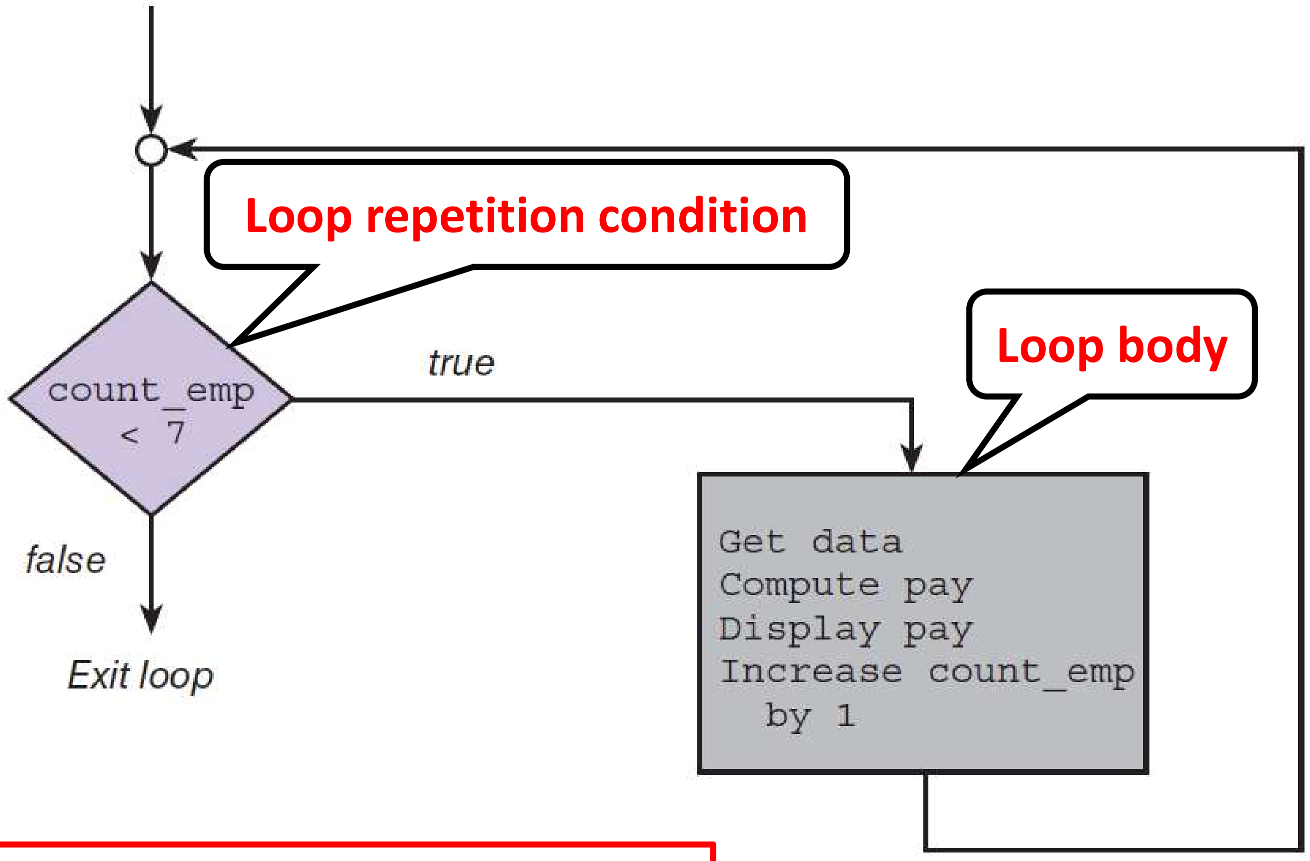
Example of a `while` Loop

- Compute and display the gross pay for 7 employees
  - Initialization: `count_emp = 0;`
  - Testing: `(count_emp < 7)`
  - Updating: `count_emp = count_emp + 1;`



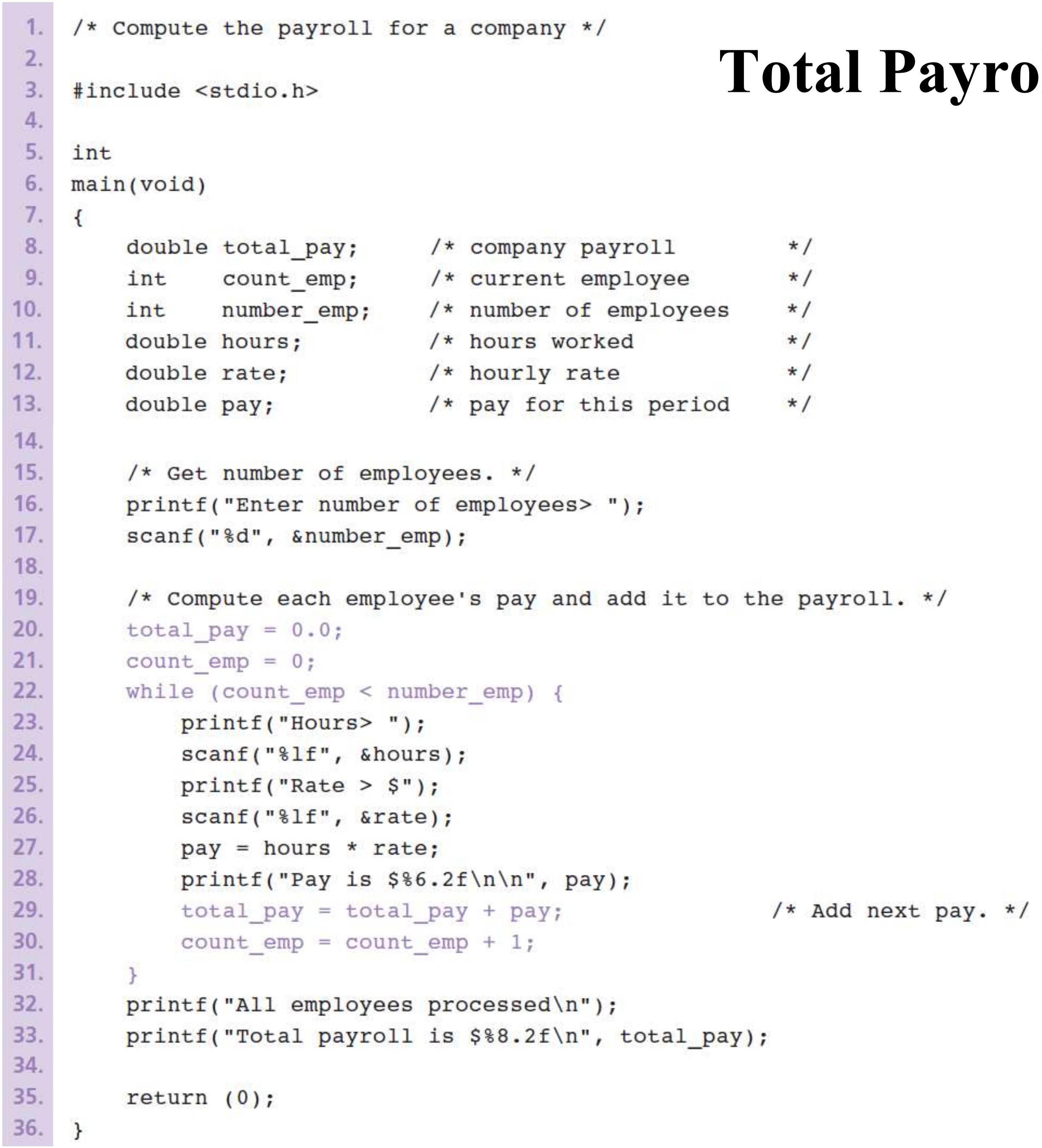
```
count_emp = 0;              /* no employees processed yet     */
while (count_emp < 7) {     /* test value of count_emp        */
    printf("Hours> ");
    scanf("%d", &hours);
    printf("Rate> ");
    scanf("%lf", &rate);
    pay = hours * rate;
    printf("Pay is $%6.2f\n", pay);
    count_emp = count_emp + 1; /* increment count_emp         */
}
printf("\nAll employees processed\n");
```

Flowchart of a `while` Loop



If **count_emp** is not updated, the loop will execute forever. Such a loop is called **infinite loop**.

Total Payroll of a Company



```c
/* Compute the payroll for a company */

#include <stdio.h>

int
main(void)
{
    double total_pay;     /* company payroll         */
    int    count_emp;     /* current employee        */
    int    number_emp;    /* number of employees     */
    double hours;         /* hours worked            */
    double rate;          /* hourly rate             */
    double pay;           /* pay for this period     */

    /* Get number of employees. */
    printf("Enter number of employees> ");
    scanf("%d", &number_emp);

    /* Compute each employee's pay and add it to the payroll. */
    total_pay = 0.0;
    count_emp = 0;
    while (count_emp < number_emp) {
        printf("Hours> ");
        scanf("%lf", &hours);
        printf("Rate > $");
        scanf("%lf", &rate);
        pay = hours * rate;
        printf("Pay is $%6.2f\n\n", pay);
        total_pay = total_pay + pay;               /* Add next pay. */
        count_emp = count_emp + 1;
    }
    printf("All employees processed\n");
    printf("Total payroll is $%8.2f\n", total_pay);

    return (0);
}
```

Sample Run

```
Enter number of employees> 3
Hours> 50
Rate> $5.25
Pay is $262.50

Hours> 6
Rate> $5.0
Pay is $ 30.00

Hours> 15
Rate> $7.0
Pay is $105.00

All employees processed
Total payroll is $ 397.50
```

Sum of numbers using while Loop

```c
#include<stdio.h>

int main(void)                /* Compute Sum of numbers */
{
    int i = 0;               /* count number */
    int a;                   /* current input number */
    int sum = 0;             /* Sum of inputs */

    while(i < 10)
    {
        printf("Enter a number: ");
        scanf("%d", &a);
        sum = sum + a;
        i++;
    }
    printf("Total is %d\n", sum);

    return 0;
}
```

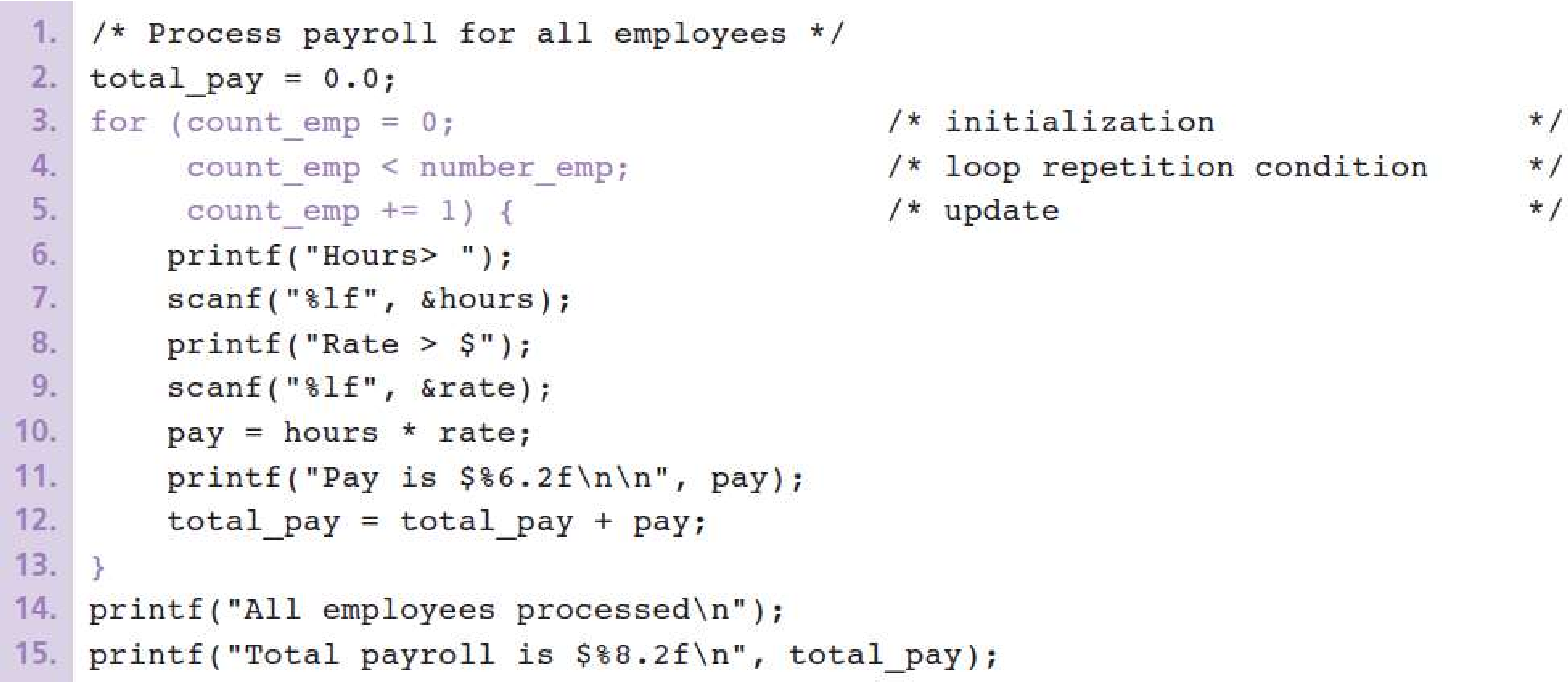
The for Statement

- Better way to write a counting loop

```
for (initialization expression;
     loop repetition condition;
     update expression)
     Statement ; /* Can be Compound */
```

- First, the initialization expression is executed
- Then, the loop repetition condition is tested
  - If true, the Statement is executed , the update expression is computed, and the repetition condition is re-tested
- Repeat as long as the repetition condition is true

Accumulating a Sum: total_pay



```
/* Process payroll for all employees */
total_pay = 0.0;
for (count_emp = 0;                      /* initialization                */
     count_emp < number_emp;             /* loop repetition condition     */
     count_emp += 1) {                   /* update                        */
    printf("Hours> ");
    scanf("%lf", &hours);
    printf("Rate > $");
    scanf("%lf", &rate);
    pay = hours * rate;
    printf("Pay is $%6.2f\n\n", pay);
    total_pay = total_pay + pay;
}
printf("All employees processed\n");
printf("Total payroll is $%8.2f\n", total_pay);
```

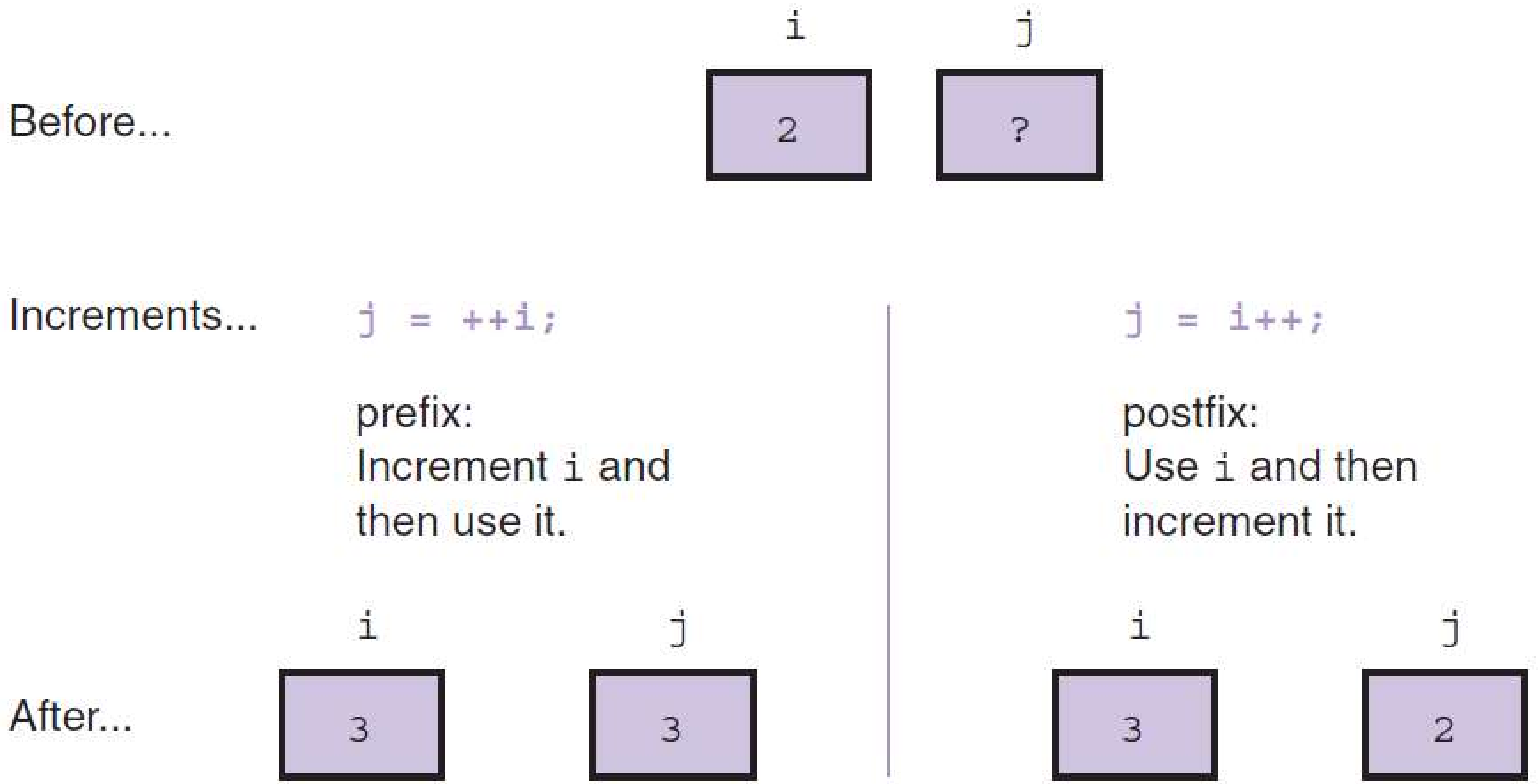
Compound Assignment Operator

`variable op= expression ;` *is equivalent to*

`variable = variable op (expression) ;`

Statement with Simple Assignment Operator	Equivalent with Compound Assignment Operator
`count_emp = count_emp + 1;`	`count_emp += 1;`
`time = time - 1;`	`time -= 1;`
`product = product * item;`	`product *= item;`
`total = total / number;`	`total /= number;`
`n = n % (x+1);`	`n %= x+1;`

Prefix and Postfix Increments



Before...

Increments...

`j = ++i;`

prefix:
Increment `i` and
then use it.

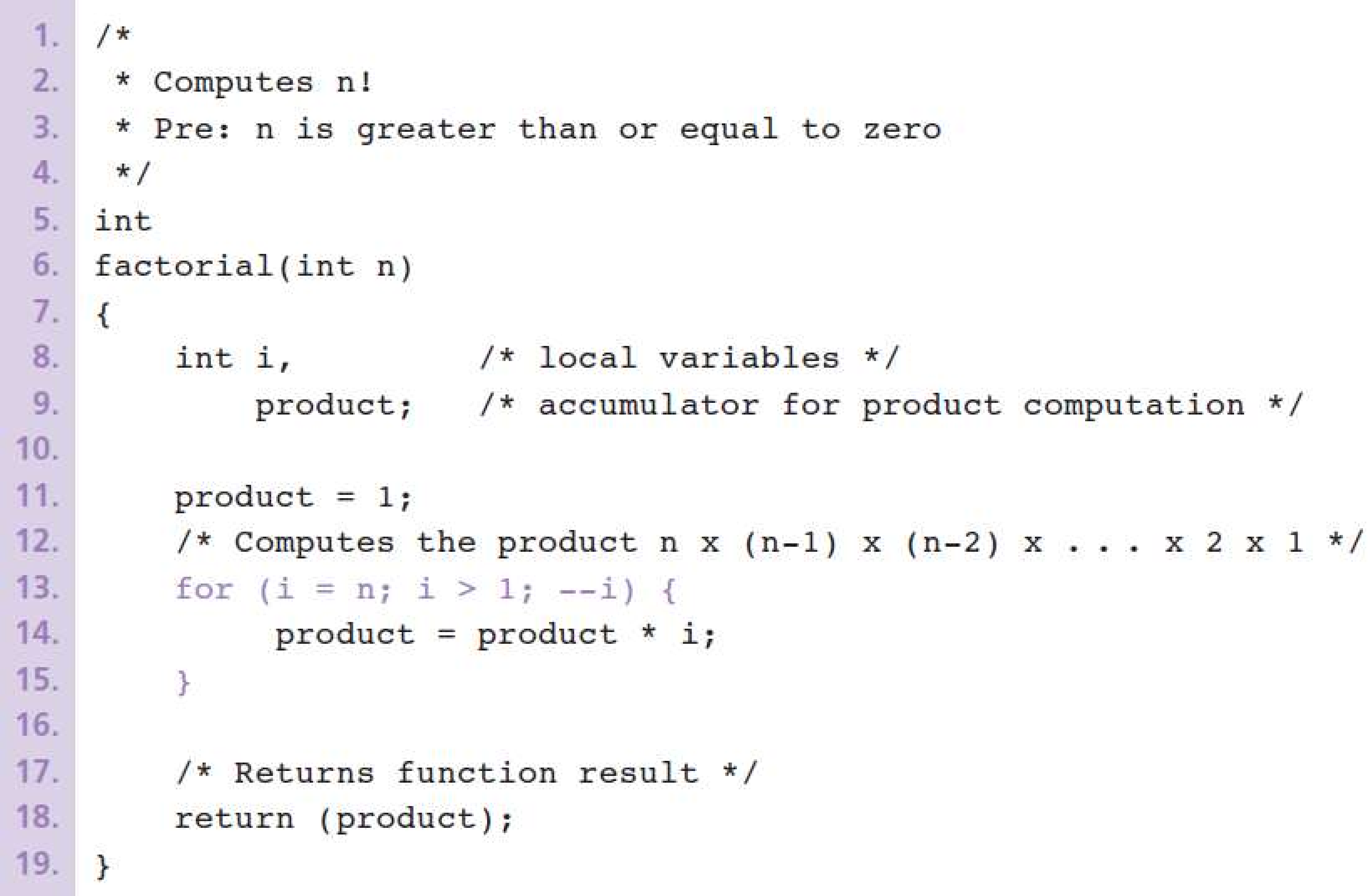
`j = i++;`

postfix:
Use `i` and then
increment it.

After...

C also provides the **decrement** operator **--** that can be used in either the prefix or postfix position

Computing Factorial

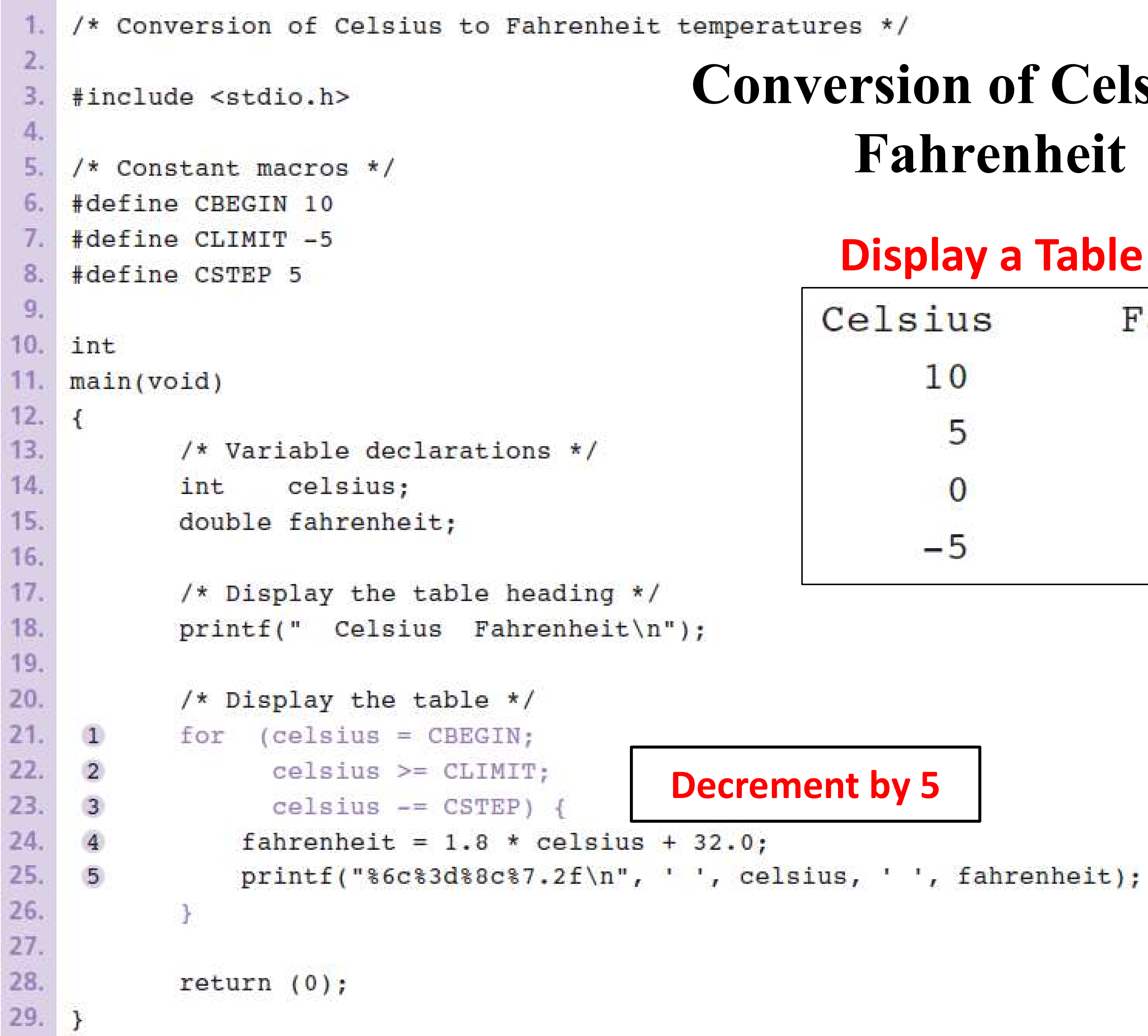


```
/*
 * Computes n!
 * Pre: n is greater than or equal to zero
 */
int
factorial(int n)
{
    int i,         /* local variables */
        product;   /* accumulator for product computation */

    product = 1;
    /* Computes the product n x (n-1) x (n-2) x . . . x 2 x 1 */
    for (i = n; i > 1; --i) {
         product = product * i;
    }

    /* Returns function result */
    return (product);
}
```

Conversion of Celsius to Fahrenheit



```c
/* Conversion of Celsius to Fahrenheit temperatures */

#include <stdio.h>

/* Constant macros */
#define CBEGIN 10
#define CLIMIT -5
#define CSTEP 5

int
main(void)
{
       /* Variable declarations */
       int    celsius;
       double fahrenheit;

       /* Display the table heading */
       printf("  Celsius  Fahrenheit\n");

       /* Display the table */
       for  (celsius = CBEGIN;
             celsius >= CLIMIT;
             celsius -= CSTEP) {
           fahrenheit = 1.8 * celsius + 32.0;
           printf("%6c%3d%8c%7.2f\n", ' ', celsius, ' ', fahrenheit);
       }

       return (0);
}
```

Display a Table of Values

```
Celsius     Fahrenheit
     10          50.00
      5          41.00
      0          32.00
     -5          23.00
```

Decrement by 5

Conditional Loop

- Not able to determine the exact number of loop repetitions before loop execution begins
- Example of a conditional loop: **input validation**

```
printf("Enter number of students> ");
scanf("%d", &num_students);
while (num_students < 0) {
  printf("Invalid negative number; try again> ");
  scanf("%d", &num_students);
}
```

- `while` loop rejects invalid (negative) input

Sentinel-Controlled Loop

- In many programs, we input a list of data values
- Often, we don't know the length of the list
- We ask the user to enter a unique data value, called a sentinel value, after the last data item
- **Sentinel Value**
  - An end marker that follows the last value in a list of data
  - For readability, we used **`#define`** to name the **`SENTINEL`**
- The loop repetition condition terminates a loop when the sentinel value is read

Sentinel-Controlled `while` Loop

```c
#include <stdio.h>
#define SENTINEL -1  /* Marking end of input */

int main(void) {     /* Compute the sum of test scores */
  int sum = 0;       /* Sum of test scores */
  int score;         /* Current input score */

  printf("Enter first score (%d to quit)> ", SENTINEL);
  scanf("%d", &score);
  while (score != SENTINEL) {
    sum += score;
    printf("Enter next score (%d to quit)> ", SENTINEL);
    scanf("%d", &score);
  }
  printf("\nSum of exam scores is %d\n", sum);
  return (0);
}
```

Sentinel-Controlled for Loop

```
#include <stdio.h>
#define SENTINEL -1  /* Marking end of input */

int main(void) {     /* Compute the sum of test scores */
  int sum = 0;       /* Sum of test scores */
  int score;         /* Current input score */

  printf("Enter first score (%d to quit)> ", SENTINEL);
  for (scanf("%d", &score);
       score != SENTINEL;
       scanf("%d", &score)) {
    sum += score;
    printf("Enter next score (%d to quit)> ", SENTINEL);
  }
  printf("\nSum of exam scores is %d\n", sum);
  return (0);
}
```

Infinite Loop on Faulty Input Data

- Reading faulty data can result in an infinite loop

```
scanf("%d", &score); /* read integer */
```

- Suppose the user enters the letter `X`

```
Enter next score (-1 to quit)> X
```

`scanf` fails to read variable `score` as letter `X`

- Variable `score` is **not modified** in the program

`score != SENTINEL` is always **true**

- Therefore, **Infinite Loop**

Detecting Faulty Input Data

- `scanf` can detect faulty input as follows:

```
status = scanf("%d", &score);
```

- If **`scanf`** successfully reads **`score`** then **`status`** is **`1`**
- If **`scanf`** fails to read **`score`** then **`status`** is **`0`**
- We can test **`status`** to detect faulty input
- This can be used to terminate the execution of a loop
- In general, **`scanf`** can read multiple variables
- It returns the number of successfully read inputs

Terminating Loop on Faulty Input

```
int main(void) {     /* Compute the sum of test scores */
  int sum = 0;       /* Sum of test scores */
  int score;         /* Current input score */
  int status;        /* Input status of scanf */

  printf("Enter first score (%d to quit)> ", SENTINEL);
  status = scanf("%d", &score);
  while (status != 0 && score != SENTINEL) {
    sum += score;
    printf("Enter next score (%d to quit)> ", SENTINEL);
    status = scanf("%d", &score);
  }
  printf("\nSum of exam scores is %d\n", sum);
  return (0);
}
```

Print number in reverse order

```
#include <stdio.h>

int main(void)
{
    int number, digit;

    printf("Enter a number: ");
    scanf("%d", &number);

    while(number > 0)
    {
        digit = number % 10;
        printf("%d", digit);
        number = number / 10;
    }

    return 0;
}
```

Nested Loops

- Consist of an outer loop with one or more inner loops
- Each time the outer loop is repeated, the inner loops are reentered and executed
- **Example:**

outer loop

inner loop

```
int n = 5;
int i, j;
 for (i = 1; i <= n; i++)
 {
  for (j = 1; j <= i; j++)
  {
   printf("*");
  }
  printf("\n");
 }
```

```
*
**
***
****
*****
```

What is the Output

```
/* Illustrates nested for loops */
#include <stdio.h>

int  main(void) {
  int i, j; /* loop variables */
  printf("          I    J\n");
  for (i = 1; i < 4;  i++) {
    printf("Outer %6d\n", i);
    for (j = 0; j < i; j++) {
      printf("  Inner%9d\n", j);
     }   /* end of inner loop */
   }     /* end of outer loop */

   return (0);
}
```

```
          i   j
Outer     1
  Inner       0
Outer     2
  Inner       0
  Inner       1
Outer     3
  Inner       0
  Inner       1
  Inner       2
```

The `do-while` Statement

- The **`for`** and **`while`** statements evaluate the loop condition **before** the execution of the loop body
- The **`do-while`** statement evaluates the loop condition **after** the execution of the loop body
- **Syntax:**

```
do
  statement; /* Can be compound */
while (loop repetition condition) ;
```

- The **`do-while`** must execute **at least one time**

Using do-while to repeat Program

```
int main(void) {
  . . .           /* Variable Declarations */
  char ch;        /* User response [y/n] */

  do {
    . . .               /* Execute program */
    printf("Repeat again [y/n]? ");
    ch = getch();       /* read from keyboard */
    printf("%c\n", ch); /* display character */
  } while (ch=='y'|| ch=='Y’);
  return 0;
}
```

Example: Selection Inside Loop

```
#include<stdio.h>

int main(void)
{
    int number, i, flag = 1;
    scanf("%d", &number);

    for(i = 2; i < number; i++)
    {
        if(number % i == 0)
            flag = 0;
    }

    if(flag == 1)
        printf("%d is a prime number.\n", number);
    else
        printf("%d is not a prime number.\n", number);

    return 0;
}
```

Using break Inside Loop

```
#include<stdio.h>

int main(void){
    int number, i, flag = 1;
    scanf("%d", &number);

    for(i = 2; i < number; i++){
        if(number % i == 0){
            flag = 0;
            break;
        }
    }

    if(flag == 1)
        printf("%d is a prime number",number);
    else
        printf("%d is not a prime number",number);

    return 0;
}
```

The **break** statement makes the loop terminate prematurely.

Using `continue` Inside Loop

```c
#include<stdio.h>

int main(void){
    int number, i, sum = 0;

    for(i = 0; i < 10; i++){
        printf("Enter a number: ");
        scanf("%d", &number);

        if(number < 0)
            continue;

        sum += number;
        printf("%d is added\n", number);
    }

    printf("Total = %d\n",sum);
    return 0;
}
```

The **continue** statement forces next iteration of the loop, skipping any remaining statements in the loop

Using continue Inside Loop

```
#include<stdio.h>

int main(void){
    int number, i, sum = 0;

    for(i = 0; i < 10; i++){
        printf("Enter a number: ");
        scanf("%d", &number);

        if(number < 0)
            continue;

        sum += number;
        printf("%d is added\n", number);
    }

    printf("Total = %d\n",sum);
    return 0;
}
```

The **continue** statement forces next iteration of the loop, skipping any remaining statements in the loop

Using continue Inside Loop

```c
#include<stdio.h>

int main(void){
    int number, i, sum = 0;

    for(i = 0; i < 10; i++){
        printf("Enter a number: ");
        scanf("%d", &number);

        if(number < 0)
            continue;

        sum += number;
        printf("%d is added\n", number);
    }

    printf("Total = %d\n",sum);
    return 0;
}
```

Output:

```
Enter a number: 1
1 is added
Enter a number: 2
2 is added
Enter a number: 3
3 is added
Enter a number: -4
Enter a number: -5
Enter a number: 6
6 is added
Enter a number: 7
7 is added
Enter a number: 8
8 is added
Enter a number: -9
Enter a number: 10
10 is added
Total = 37
```

How to Debug and Test a Program

- Using a debugger program
  - **Debug option** should be selected
  - Execute program one statement at a time (**Next line**)
  - Watch the value of variables at runtime (**Add watch**)
  - Set **breakpoints** at selected statements
- Debugging without a debugger
  - Insert ***extra printf statements*** that display intermediate results at critical points in your program

    ```
    if (DEBUG) printf(. . .);
    ```
  - Turn ON diagnostic calls to `printf`

    ```
    #define DEBUG 1
    ```

Example: Debugging using printf

```
#define DEBUG 1  /* turn on diagnostics */
#define DEBUG 0  /* turn off diagnostics */
```

```
int main(void) {
  int score, sum=0;
  printf("Enter first score (%d to quit)> ", SENTINEL);
  scanf("%d", &score);      /* get first score */
  while (score != SENTINEL) {
    sum += score;
    if (DEBUG) printf("score=%d, sum=%d\n", score, sum);
    printf("Enter next score (%d to quit)> ", SENTINEL);
    scanf("%d", &score);   /* get next score */
  }
  printf("Total score is %d\n", sum);
  return 0;
}
```

Off-By-One Loop Errors

- A common logic error
- A loop executes one more time or one less time
- **Example:**

```
for (count = 0; count <= n; ++count)
  sum += count;
```

Executes n + 1 times

```
for (count = 1; count < n; ++count)
  sum += count;
```

Executes n – 1 times

- Checking loop boundaries
  - Initial and final values of the loop control variable

Common Programming Errors

- Do not confuse **`if`** and **`while`** statements
  - **`if`** statement implements a decision step
  - **`while`** statement implements a loop
- **`for`** loop: remember to end the initialization step and the loop repetition condition with **semicolon (;)**
- Remember to use **braces { and }** around a loop body consisting of multiple statements
- Remember to provide a **prompt** for the user, when using a sentinel-controlled loop
- Make sure the sentinel value cannot be confused with a normal data input

Common Programming Errors

- Use `do-while` only when there is no possibility of zero loop iterations
- Do not use increment, decrement, or compound assignment as sub-expressions in complex expressions

```
a *= b + c;        /* a = a*(b+c); */
```

There is no shorter way to write: `a = a*b + c;`

- Be sure that the operand of an increment/decrement operator is a variable:

```
z = ++j * k--;  /* ++j; z=j*k; k--; */
```

Chapter Review

- Two kinds of loops occur frequently in programming
- **Counting loop:** controlled by a counter
- **Conditional loop:** controlled by a condition
  - **Sentinel-controlled loop**
  - **Input validation loop**
  - **General conditional loop**
- C provides three statements for implementing loops
  - `while` statement (can have zero repetitions)
  - `for` statement (can have zero repetitions)
  - `do-while` statement (must execute at least once)